Security Analysis of CPace

Michel Abdalla
CNRS and DI/ENS, PSL University
Paris, France
michel.abdalla@ens.fr

Björn Haase
Endress+Hauser Liquid Analysis
Gerlingen, Germany
bjoern.haase@endress.com

Julia Hesse
IBM Research
Zurich, Switzerland
jhs@zurich.ibm.com

ABSTRACT

In response to standardization requests regarding password-authenticated key exchange (PAKE) protocols, the IRTF working group CFRG has setup a PAKE selection process in 2019, which led to the selection of the CPace protocol in the balanced setting, in which parties share a common password.

In this paper, we provide a security analysis of CPace in the universal composability framework for implementations on elliptic-curve groups. When doing so, we restrict the use of random oracles to hash functions only and refrain from modeling CPace's Map2Point function that maps field elements to curve points as an idealized function. As a result, CPace can be proven secure under standard complexity assumptions in the random-oracle model.

Finally, in order to extend our proofs to different CPace variants optimized for specific environments, we employ a new approach, which represents the assumptions required by the proof as libraries which a simulator can access. By allowing for the modular replacement of assumptions used in the proof, this new approach avoids a repeated analysis of unchanged protocol parts and lets us efficiently analyze the security guarantees of all the different CPace variants.

1 INTRODUCTION

Security analysis and efficient implementation of cryptographic protocols are often split into separate working groups. As a result, subtle differences between the actually implemented and analyzed protocols easily emerge, for example when implementors slightly tweak the protocol to improve efficiency. An example where particularly aggressive optimization for efficiency is implemented on the protocol level is CPace as specified in current internet drafts [Haa20, HL18]. CPace is a password-authenticated key exchange protocol (PAKE) [BM92], which allows two parties to establish a shared cryptographic key from matching passwords of potentially low entropy. On a high level, CPace works as follows. Given a cyclic group $\mathcal{G}$, parties first locally and deterministically compute a generator $g \in \mathcal{G}$ from their passwords in a secure way, so that g reveals as little information about the password as possible. Then, both parties perform a Diffie-Hellman key exchange by choosing secret exponents x and y, respectively, exchanging g^x and g^y and locally compute $K = (g^x)^y = (g^y)^x$. The final key is then computed as the hash of K together with session-identifying information such as transcript. The currently most efficient implementations of the above blueprint protocol use elliptic curve groups of either prime or composite order. To securely compute the generator, the password is first hashed to the finite field $\mathbb{F}_q$ the curve is constructed over, and then mapped to the curve by a map called $\mathsf{Map2Point}$. Depending on the choice of curve, efficiency tweaks such as simplified point verification on curves with twist security, or computation with only x-coordinates of points can be applied [HL19, Haa20]. Unfortunately, until today, it is not clear how these modifications impact security of CPace.

A short history of CPace. In 1996, Jablon [Jab96] introduced the SPEKE protocol, which performs a Diffie-Hellman key exchange with generators computed as $g \leftarrow \mathsf{H}(\mathsf{pw})$. Many variants of SPEKE have emerged in the literature since then, including ones that fixed initial security issues of SPEKE. Among them, the PACE protocol [PAC08, BFK09] aims at circumventing direct hashing onto the group with an interactive $\mathsf{Map2Point}$ protocol to compute the password-dependent generators. From this, CPace [HL19] emerged by combining the best properties of PACE and SPEKE, namely computing the generator without interaction while avoiding the need to hash directly onto the group. More precisely, password-dependent generators are computed as $g \leftarrow \mathsf{Map2Point}(\mathsf{H}(\mathsf{pw}))$. In 2020, the IRTF working group CFRG has chosen CPace as the recommended protocol for (symmetric) PAKE.

Our Contributions. In this paper, we provide the first comprehensive security analysis of the CPace protocol that applies also to variants of CPace optimized for usage with state-of-the-art elliptic curves. On a technical level, in Section 3.3 we first identify the core properties of the deterministic $\mathsf{Map2Point}$ function that allow to prove strong security properties of CPace. Crucially, we restrict the use of random oracles to hash functions only and refrain from modeling $\mathsf{Map2Point}$ as an idealized function, as it would not be clear how to instantiate it in practice. We show that, using some weak invertibility properties of $\mathsf{Map2Point}$ that we demonstrate to hold for many candidate implementations, CPace can be proven secure under standard Diffie-Hellman-type assumptions in the random-oracle model. Our security proof captures adaptive corruptions and weak forward secrecy and is carried out in the Universal Composability (UC) framework, which is today's standard when analyzing security of PAKE protocols.

In Section 6 we then turn our attention to modifications of CPace and, for each modification individually, state under which assumptions the security properties are preserved. In more detail, our analysis captures the following modifications.

- Using groups of composite order $c \cdot p$, where p is a large prime and c is a small cofactor coprime to p.
- Consider both, constructions based on a map-twice-and-add strategy and single execution of $\mathsf{Map2Point}$.
- Using only single-coordinate representations of elliptic curve points in order to speed up and facilitate implementation.
- Avoiding computationally costly point verification on curves with secure quadratic twists such as Curve25519[Ber06].

To demonstrate the security of these variants, we take a new approach that saves us from a repeated analysis of unchanged parts of CPace. Namely, we implement the CDH-type cryptographic assumptions required by CPace as libraries which a simulator can access. This allows for modular replacement of assumptions required in the security proof, and lets us efficiently analyze all the different CPace variants' security guarantees. We believe that this new proof technique introduced in Section 5.1 might be of independent interest in particular for machine-assisted proving, since reductions are captured in code instead of textual descriptions only.

As a side contribution, we identify a common shortcoming in all UC PAKE security definitions in the literature [CHK+05, JKX18, Hes20, ABB+20a], which may impact the usability of these definitions as building blocks in higher-level applications. All these definitions allow a malicious party to learn the shared key computed by an honest party *without knowing her password*. We strengthen the definition to prevent such attacks, and demonstrate with our analysis of CPace that our fix yields a security definition that is still met by PAKE protocols. Even more, our analysis captures adaptive corruptions, where an adversary can corrupt users who already started the protocol and computed secret values. Our work provides the first evidence that SPEKE-like protocols can withstand these strong corruptions.

In conclusion, our results demonstrate that CPace enjoys strong provable security guarantees in all its variants that have been proposed for the sake of efficiency improvements.

Related Work. In [BFK09] a game-based security analysis of explicitly authenticated PACE protocol variants as used in travel documents has been carried out, with a focus on different variants of *interactive* Map2Point constructions. Security of CPace (including Map2Point and cofactor clearing) was formally analyzed in [HL19]. However, the proof was found to be insufficient by reviews done during the CFRG selection process, and indeed, the claimed security under plain CDH seems to be difficult to achieve. Further, no analysis of real world implementation artifacts such as cofactor clearing, map2point function and twist security was conducted in that work. Altogether, for establishing full trust in real-world implementations of CPace we identify the urgent need for filling the gaps in the existing analysis in [HL19].

[ABB+20a] analyzed the security of several EKE [BM92] and SPEKE variants in the UC framework, including SPAKE2 [AP05] and TBPEKE [PW17]. They also indicated that their proof for TBPEKE could be extended to CPace with the inclusion of the protocol transcript and password-dependent generator in the final key derivation hash [ABB+20b]. Although the need for the password-dependent generator in the final hash could be excluded with a more refined argument, their analysis still does not deal with adaptive security and only considers a simplified version of CPace, similar to the CPace$_{\text{base}}$ variant in Section 5, which does not account for the implementation issues being addressed in this paper.

[ABK+20] formalized the algebraic-group model within the UC framework and proved that the SPAKE2 and CPace protocols are universally composable in the new model with respect to the standard functionality for password-based authenticated key exchange in [CHK+05]. As in [ABB+20a], their analysis does not deal with adaptive security and only considers a simplified version of CPace, similar to CPace$_{\text{base}}$.

Organization of this paper. We introduce the PAKE security model in Section 2, hardness assumptions and requirements for the map in Section 3. We then repeat the essential properties of the CPace construction in Section 4. Then we analyse CPace, first using a simplified CPace in Section 5 (modeling the map as random-oracle) and then extending the analysis to real-world instantiations using mapping constructions in Section 6. Section 7 presents some concluding remarks and possible directions for future work.

Session initiation

On (NewSession, $sid, \mathcal{P}, \mathcal{P}'$, pw) from $\mathcal{P}$, ignore this query if record $\langle sid, \mathcal{P}, \cdot, \cdot \rangle$ already exists. Otherwise record $\langle sid, \mathcal{P}, \mathcal{P}', \text{pw} \rangle$ marked fresh and send (NewSession, $sid, \mathcal{P}, \mathcal{P}'$) to $\mathcal{A}$.

Active attack

- On (TestPwd, $sid, \mathcal{P}$, pw*) from $\mathcal{A}$, if $\exists$ a fresh record $\langle sid, \mathcal{P}, \mathcal{P}', \text{pw}, \cdot \rangle$ then:
  - If pw* = pw then mark it compromised and return "correct guess";
  - If pw$^* \neq$ pw then mark it interrupted and return "wrong guess".
- On (RegisterTest, $sid, \mathcal{P}$) from $\mathcal{A}$, if $\exists$ a fresh record $\langle sid, \mathcal{P}, \mathcal{P}', \cdot \rangle$ then mark it interrupted and flag it tested.
- On (LateTestPwd, $sid, \mathcal{P}$, pw*) from $\mathcal{A}$, if $\exists$ a record $\langle sid, \mathcal{P}, \mathcal{P}', \text{pw}, K \rangle$ marked completed with flag tested then remove this flag and do:
  - If pw* = pw then return $\boxed{K}$ to $\mathcal{A}$;
  - If pw$^* \neq$ pw then return $\boxed{K^{\$} \leftarrow_{\text{R}} \{0,1\}^{\kappa}}$ to $\mathcal{A}$.

Key generation

On (NewKey, $sid, \mathcal{P}, K^*$) from $\mathcal{A}$, if $\exists$ a record $\langle sid, \mathcal{P}, \mathcal{P}', \text{pw} \rangle$ not marked completed then do:

- If the record is compromised, or either $\mathcal{P}$ or $\mathcal{P}'$ is corrupted, then set $K := K^*$.
- If the record is fresh and $\exists$ a completed record $\langle sid, \mathcal{P}', \mathcal{P}, \text{pw}, K' \rangle$ that was fresh when $\mathcal{P}'$ output (sid, K'), then set $K := K'$.
- In all other cases pick $K \leftarrow_{\text{R}} \{0,1\}^{\kappa}$.

Finally, append K to record $\langle sid, \mathcal{P}, \mathcal{P}', \text{pw} \rangle$, mark it completed, and output (sid, K) to $\mathcal{P}$.

Adaptive corruption

On (AdaptiveCorruption, $sid, \mathcal{P}$) from $\mathcal{A}$, if $\exists$ a record $\langle sid, \mathcal{P}, \cdot, \text{pw} \rangle$ not marked completed then mark it completed and output (sid, pw).

Figure 1: UC PAKE variants: The original PAKE functionality $\mathcal{F}_{\text{pwKE}}$ of Canetti et al. [CHK+05] is the version with all gray text omitted. The lazy-extraction PAKE functionality $\mathcal{F}_{\text{lePAKE}}$ [ABB+20a] includes everything, and the variant of $\mathcal{F}_{\text{lePAKE}}$ used in this work includes everything but the dashed box.

2 PAKE SECURITY MODEL

We use the Universal Composability (UC) framework of Canetti [Can01] to formulate security properties of CPace. For PAKE, usage of the simulation-based UC framework comes with several advantages over the game-based model for PAKE introduced by Bellare et al. [BPR00]. Most importantly, UC secure PAKE protocols preserve their security properties in the presence of adversarially-chosen passwords and when composed with arbitrary other protocols. Originally introduced by Canetti et al. [CHK+05], the ideal functionality $\mathcal{F}_{\text{pwKE}}$ for PAKE (depicted in Fig. 1) is accessed by two parties, $\mathcal{P}$ and $\mathcal{P}'$, who both provide their passwords. $\mathcal{F}_{\text{pwKE}}$ then provides both parties with a uniformly random session key if passwords match, and with individual random keys if passwords mismatch. Since an adversary can always engage in a session and guess the counterpart's password with non-negligible probability, $\mathcal{F}_{\mathsf{pwKE}}$ must include an adversarial interface TestPwd for such guesses. Crucially, only one guess against every honest party is allowed, modeling the fact that password guessing is an online attack and cannot be used to brute-force the password from a protocol's transcript. We refer the reader to [CHK+05] for a more comprehensive introduction to the PAKE functionality.

An ideal functionality for the SPEKE protocol family. Unfortunately, $\mathcal{F}_{\mathsf{pwKE}}$ is not suitable to analyze SPEKE-like PAKE protocols such as CPace, where session keys are computed as hashes of Diffie-Hellman keys (and possibly parts of the transcript). The reason is that $\mathcal{F}_{\mathsf{pwKE}}$'s TestPwd interface allows password guesses only *during* a protocol run, which requires a simulator to extract password guesses from the protocol's transcript. When the final output is a hash, the adversary might postpone its computation, keeping information from the simulator that is required for password extraction. To circumvent these issues, recently a "lazy-extraction PAKE" functionality $\mathcal{F}_{\mathsf{lePAKE}}$ was proposed and shown useful in the analysis of SPEKE-like protocols by Abdalla et al. [ABB+20a]. $\mathcal{F}_{\mathsf{lePAKE}}$, which we also depict in Fig. 1, allows *either* one online *or* one offline password guess after the key exchange was finished. One might argue that usage of keys obtained from $\mathcal{F}_{\mathsf{lePAKE}}$ is never safe, since the adversary might eventually extract the key from it at any later point in time. This however can be easily prevented by adding a key confirmation round, which keeps an adversary from postponing the final hash query and guarantees perfect forward secrecy [ABB+20a]. We refer the reader to [ABB+20a] for a thorough discussion of $\mathcal{F}_{\mathsf{lePAKE}}$.

Our adjustments to $\mathcal{F}_{\mathsf{lePAKE}}$. The main difference between our $\mathcal{F}_{\mathsf{lePAKE}}$ and all PAKE functionalities from the literature [CHK+05, JKX18, Hes20, ABB+20a] is that we remove a shortcoming that rendered these functionalities essentially useless as building blocks for higher-level applications. More detailed, we remove the ability of the adversary to determine an honest party's output key in a corrupted session. The change can be seen in Fig. 1, where the dashed box shows the weakening that we simply omit in our version of $\mathcal{F}_{\mathsf{lePAKE}}$. In reality, nobody would want to use a PAKE where an adversary can learn (even set) the key of an honest party *without knowing the honest party's password.* This is not what one would expect from an authenticated key exchange protocol. In Appendix B we explain why existing PAKE protocols can still be considered secure, but also provide an illustrating example how this shortcoming hinders usage of PAKE functionalities in modular protocol analysis. In this paper, we demonstrate that CPace can be proven to protect against such attacks.

We also make two minor adjustments, which are merely to ease presentation in this paper. Namely, we add an explicit interface for adaptive corruptions, and we omit roles since we analyze a protocol where there is no dedicated initiator.

3 PRELIMINARIES

3.1 Notation

Throughout the paper, we use k as security parameter. With $\leftarrow_{\mathrm{R}}$ we denote uniformly random sampling from a set. With $\mathrm{oc}(P_i, P_j)$ we denote ordered concatenation, i.e., $\mathrm{oc}(P_i, P_j) = P_i||P_j$ if $P_i \leq P_j$ and $\mathrm{oc}(P_i, P_j) = P_j||P_i$ otherwise.

3.2 Cryptographic assumptions

The security of CPace is based on the hardness of a combination of strong and simultaneous Diffie-Hellman problems. To ease access to the assumptions, we state them with increasing complexity.

Definition 3.1 (Strong CDH problem (sCDH) [ABR01]). *Let $\mathcal{G}$ be a cyclic group with a generator B and $(Y_a = B^{y_a}, Y_b = B^{y_b})$ sampled uniformly from $(\mathcal{G} \backslash I_{\mathcal{G}})^2$. Given access to oracles DDH $(B, Y_a, \cdot, \cdot)$ and DDH $(B, Y_b, \cdot, \cdot)$, provide K such that $K = B^{y_a \cdot y_b}$.*

We note that sCDH is a weaker variant of the so-called gap-CDH assumption, where the adversary has access to "full" DDH oracles with no fixed inputs. Next we provide a stronger variant of sCDH where two CDH instances need to be solved that involve a common, adversarially chosen element.

Definition 3.2 (Strong simultaneous CDH problem (sSDH)). *Let $\mathcal{G}$ be a cyclic group and (Y_a, G_1, G_2) sampled uniformly from $(\mathcal{G} \setminus I_{\mathcal{G}})^3$. Given access to oracles DDH $(G_1, Y_a, \cdot, \cdot)$ and DDH $(G_2, Y_a, \cdot, \cdot)$, provide $(X_b, K_1, K_2) \in (\mathcal{G} \setminus I_{\mathcal{G}}) \times \mathcal{G} \times \mathcal{G}$ s. th. DDH (G_1, Y_a, X_b, K_1) = DDH $(G_2, Y_a, X_b, K_2) = 1$*

Again we note that sSDH is a weaker variant of the gap simultaneous Diffie-Hellman problem from [PW17] and [ABB+20a] as the access to the DDH oracles is restricted by fixing two inputs. Lastly, we state a variant of the sSDH assumption where generators are sampled according to some probability distribution. Looking ahead, we require this variant since in CPace parties derive generators by applying a map which does not implement uniform sampling from the group. We state the non-uniform variant of sSDH for arbitrary probability distributions and investigate its relation to "uniform" sSDH afterwards.

Definition 3.3 (Strong simultaneous non-uniform CDH problem ($\mathcal{D}_{\mathcal{G}}$-sSDH)). *Let $\mathcal{G}$ be a group and $\mathcal{D}_{\mathcal{G}}$ be a probability distribution on $\mathcal{G}$. The strong simultaneous non-uniform CDH problem $\mathcal{D}_{\mathcal{G}}$-sSDH is defined as the sSDH problem but with (Y_a, G_1, G_2) sampled using $\mathcal{U}_{\mathcal{G}} \times \mathcal{D}_{\mathcal{G}} \times \mathcal{D}_{\mathcal{G}}$, where $\mathcal{U}_{\mathcal{G}}$ denotes the uniform distribution on $\mathcal{G}$.*

Clearly, $\mathcal{U}_{\mathcal{G} \backslash I_{\mathcal{G}}}$-sSDH is equivalent to sSDH. We show that hardness of uniform and non-uniform sSDH are equivalent given that the distribution allows for probabilistic polynomial time (PPT) rejection sampling, which we now formalize.

Definition 3.4 (Rejection sampling algorithm for $(\mathcal{G}, \mathcal{D}_{\mathcal{G}})$). *Let $\mathcal{G}$ be a group and $\mathcal{D}_{\mathcal{G}}$ be a probability distribution on $\mathcal{G}$. With $\mathcal{D}_{\mathcal{G}}(G)$ we denote the probability for point G. Let RS be a probabilistic algorithm taking as input elements $G \in \mathcal{G}$ and outputting $\perp$ or a value $\neq \perp$. Then RS is called a* rejection sampling algorithm for $(\mathcal{G}, \mathcal{D}_{\mathcal{G}})$ *if $Pr[RS(G) \neq \perp] = \mathcal{D}_{\mathcal{G}}(G)$ for $G \in \mathcal{G}$.*

Informally RS is a probabilistic algorithm which accepts (output different from $\perp$) or rejects (output $\perp$) a candidate point. When queried multiple times on the same input $G \in \mathcal{G}$, the probability that G will be accepted or rejected models the distribution $\mathcal{D}_{\mathcal{G}}$. Looking ahead, we define such a rejection sampler algorithm for the maps used in CPace and use this as part of our proof strategy. For avoiding that our simulator gets computationally exhausted, we require that $\mathcal{D}_{\mathcal{G}}$ allows for a PPT instantiation of a rejection sampling process using RS with a sufficiently large acceptance rate. We formalize this requirement as follows.

Definition 3.5 (Acceptance rate of a rejection sampler for $(\mathcal{G}, \mathcal{D}_{\mathcal{G}})$). *Let $\mathcal{G}$ be a group and $\mathcal{D}_{\mathcal{G}}$ be a probability distribution on $\mathcal{G}$. Let RS be a rejection sampling algorithm for $(\mathcal{G}, \mathcal{D}_{\mathcal{G}})$. Let $G_i \in \mathcal{G}$ be a sequence of m uniformly drawn points and $r_i = RS(G_i)$. Then RS is said to have an acceptance rate of $(1/n)$ if the number of accepted points with $r_i \neq \perp$ converges to m/n when $m \longrightarrow \infty$.*

Using these definitions, we are able to prove that given some assumptions on the distribution $\mathcal{D}_{\mathcal{G}}$ hardness of sSDH and $\mathcal{D}_{\mathcal{G}}$-sSDH are equivalent up to the additional PPT computational effort generated by the rejection sampling algorithm.

Theorem 3.6 (sSDH $\iff$ $\mathcal{D}_{\mathcal{G}}$-sSDH). *Let $\mathcal{G}$ be a cyclic group of order p and $\mathcal{D}_{\mathcal{G}}$ a probability distribution on $\mathcal{G}$. If there exists a PPT rejection sampler RS for $\left(\mathcal{G}, \mathcal{D}_{\mathcal{G}}\right)$ with acceptance rate $(1/n)$ then the probability of PPT adversaries against $\mathcal{D}_{\mathcal{G}}$-sSDH and sSDH of solving the respectively other problem differs by at most $(2\mathcal{D}(I_{\mathcal{G}}) + (1/p))$ and solving sSDH with the help of a $\mathcal{D}_{\mathcal{G}}$-sSDH adversary requires at most $2n$ executions of RS on average.*

Proof. **sSDH hard $\Rightarrow$ $\mathcal{D}_{\mathcal{G}}$ − sSDH hard:** Given an adversary $\mathcal{B}_{\mathcal{D}_{\mathcal{G}}-\mathrm{sSDH}}$ against $\mathcal{D}_{\mathcal{G}}$ − sSDH with non-negligible success probability ν, we show how to construct an adversary $\mathcal{A}_{\mathrm{sSDH}}$. On receiving an sSDH-challenge (Y_a, G_1, G_2), first note that Y_a is uniformly sampled from $\mathcal{G} \setminus \{I_{\mathcal{G}}\}$. $\mathcal{A}_{\mathrm{sSDH}}$ uniformly samples $r, s \in \mathbb{Z}_p$ until $\mathrm{RS}(G_1^r) = 1$ and $\mathrm{RS}(G_2^s) = 1$, which requires $2n$ calls to RS on average. $\mathcal{A}_{\mathrm{sSDH}}$ runs $\mathcal{B}_{\mathcal{D}_{\mathcal{G}}-\mathrm{sSDH}}$ on input (Y_a, G_1^r, G_2^s). If $\mathcal{B}$ queries $\mathrm{DDH}(G_1^r, Y_a, X, L)$, $\mathcal{A}$ queries his own oracle with $\mathrm{DDH}(G_1, Y_a, X, L^{1/r})$ and relays the answer to $\mathcal{B}$ (queries G_2^s are handled analogously). On receiving (Y_b, K_1, K_2) from $\mathcal{B}_{\mathcal{D}_{\mathcal{G}}-\mathrm{sSDH}}$, $\mathcal{A}_{\mathrm{sSDH}}$ provides $(Y_b, K_1^{1/r}, K_2^{1/s})$ as solution in his sSDH experiment.

As RS is a rejection sampler for $\mathcal{D}_{\mathcal{G}}$, (Y_a, G_1^r, G_2^s) is a random $\mathcal{D}_{\mathcal{G}}$ − sSDH challenge, and thus $\mathcal{B}$ solves it with probability ν. If $\mathcal{B}$ provides a solution, then $\mathcal{A}_{\mathrm{sSDH}}$ succeeds in solving his own challenge unless G_1^r or $G_2^s = I_{\mathcal{G}}$ or $G_1^r = G_2^s$ which occurs at most with probability $(2\mathcal{D}_{\mathcal{G}}(I_{\mathcal{G}}) + 1/p)$. As RS executes in PPT, $\mathcal{A}_{\mathrm{sSDH}}$ is PPT, uses $(2n)$ calls to RS on average and succeeds with probability $\nu(1 - 2\mathcal{D}_{\mathcal{G}}(I_{\mathcal{G}}) - 1/p)$, which is non-negligible since ν is.

sSDH hard $\Rightarrow$ $\mathcal{D}_{\mathcal{G}}$ − sSDH hard: Given an adversary $\mathcal{A}_{\mathrm{sSDH}}$ against sSDH with non-negligible probability μ we show how to construct a $\mathcal{D}_{\mathcal{G}}$ − sSDH adversary $\mathcal{B}_{\mathcal{D}_{\mathcal{G}}-\mathrm{sSDH}}$. On receiving a $\mathcal{D}_{\mathcal{G}}$ − sSDH challenge (Y_a, G_1, G_2), $\mathcal{B}$ samples $r, s \in \mathbb{Z}_p \setminus 0$ and starts $\mathcal{A}_{\mathrm{sSDH}}$ on input (Y_a, G_1^r, G_2^s). DDH oracle queries are handled the same as above. On receiving (Y_b, K_1, K_2) from $\mathcal{A}_{\mathrm{sSDH}}$, $\mathcal{B}$ provides $(Y_b, K_1^{1/r}, K_2^{1/s})$ as solution to his own challenge.

If $\mathcal{A}$ is successful, then $\mathcal{B}$ succeeds unless either G_1 or $G_2 = I_{\mathcal{G}}$ or $G_1^r = G_2^s$ which occurs at most with probability $(2\mathcal{D}_{\mathcal{G}}(I_{\mathcal{G}}) + 1/p)$. Thus, $\mathcal{B}$ is a PPT adversary against $\mathcal{D}_{\mathcal{G}}$ − sSDH succeeding with non-negligible probability $\mu(1 - 2\mathcal{D}_{\mathcal{G}}(I_{\mathcal{G}}) - 1/p)$. □

Informally, the assumptions sSDH and $\mathcal{D}_{\mathcal{G}}$ − sSDH become equivalent if stepping over an element that gets accepted in the sampling process becomes sufficiently likely for a randomly drawn sequence of candidates. Secondly, the probability of accidentaly drawing the neutral element from $\mathcal{D}_{\mathcal{G}}$ needs to be negligible.

On elliptic curves groups $\mathcal{G}$ that provide a property coined *twist security* in [BL19], CPace allows for implementations with reduced computational complexity and code size under an additional assumption including the group $\mathcal{G}'$ on the curve's quadratic twist. We formalize the corresponding problem sTCDH as follows.

Definition 3.7 (Strong twist CDH problem (sTCDH)). *Let $\mathcal{G}$ be a first cyclic group with a generator B and $(Y_a = B^{y_a})$ sampled uniformly from $(\mathcal{G} \setminus I_{\mathcal{G}})$. Let $\mathcal{G}'$ be a second cyclic group. Given access to DDH oracle $DDH(B, Y_a, \cdot, \cdot)$ in $\mathcal{G}$, provide $X, Z \in \mathcal{G}' \setminus I_{\mathcal{G}'}$ with $Z = X^{y_a}$.*

3.3 The function Map2Point

The generators of the Diffie-Hellman exchange in CPace are computed via a deterministic function $\mathsf{Map2Point} : \mathbb{F}_q \longrightarrow \mathcal{G}$, either by using one single execution of Map2Point or, alternatively, by adding the results of two independent invocations of Map2Point. In both cases, security of CPace relies on Map2Point meeting the requirements from this section. Informally, we first require that Map2Point is invertible. That is, for any point on the image of the map, there must be an efficient algorithm Map2PointPreImages that outputs all preimages in $\mathbb{F}_q$. Details on how such an inversion algorithm can be efficiently implemented for various elliptic curve groups are given in [FHSS+19, BHKL13, BCI+10a, Ham20] and references therein. Secondly, the maximum number of preimages $n_{\max}$ mapping to the same element must be konwn and needs to be small. This is needed in order to construct a rejection sampling algorithm whose acceptance rate must depend on $n_{\max}$.

Definition 3.8. *Let $\mathcal{G}$ be a group of points on an elliptic curve over a field $\mathbb{F}_q$. Let Map2Point : $\mathbb{F}_q \longrightarrow \mathcal{G}$ be a deterministic function. Then Map2Point$(\cdot)$ is called* probabilistically invertible with at most $n_{\max}$ preimages *if there exists a probabilistic polynomial-time algorithm $(r_1, \ldots, r_{n(G)}) \leftarrow$ Map2PointPreImages(G) that outputs all $n(G)$ values $r_i \in \mathbb{F}_q$ s.th. $G =$ Map2Point(r_i) for all $G \in \mathcal{G}$; and $\forall G \in \mathcal{G}$, $n_{max} \geq n(G)$.*

For a map that fulfills this property an inverse probabilistic map $\mathsf{Map2Point}^{-1}$ can be defined that also serves as rejection sampling algorithm for the distribution $\mathcal{D}_{\mathcal{G}}$ that is produced by Map2Point(r) for uniformly distributed inputs $r \in \mathbb{F}_q$:

Algorithm 1 $\mathsf{Map2Point}^{-1}(\cdot)$

```
On input $G \in \mathcal{G}$: Sample $i$ uniformly from $\{1, \ldots, n_{\max}\}$; Then
obtain $m \in \{0, \ldots, n_{\max}\}$ pre-images
$(r_1, \ldots, r_m) \leftarrow$ Map2PointPreImages($G$); If $m < i$ return $(\perp)$, else
return $(r_i)$
```

Lemma 3.9. *Given a probabilistically invertible map with at most n_{max} preimages according to Definition 3.8, then Algorithm 1 is a PPT rejection sampler for $(\mathcal{G}, \mathcal{D}_{\mathcal{G}})$ according to the Definition 3.5 with average acceptance rate $(|\mathbb{F}_q|/|\mathcal{G}|)/n_{max}$.*

Proof. We first define the average number of preimages $n_{\max} \geq \bar{n} \geq 1$ as the quotient of the order of the field $\mathbb{F}_q$ and the number of points on the image of the map, i.e., $\bar{n} = |\mathbb{F}_q|/|\text{support } (\mathcal{D}_{\mathcal{G}})|$. When drawing an element G uniformly from $\mathcal{G}$, the probability that the number of preimages m for G is nonzero is given by the quotient of the order of the support of $\mathcal{D}_{\mathcal{G}}$ and the order of the group. By the definition of $\bar{n}$ above this is $|\mathbb{F}_q|/(\bar{n}|\mathcal{G}|)$.

For any point on the map with a nonzero number m of preimages, Algorithm 1 returns a result $\neq \perp$ with probability $m/n_{\max}$. As the average value for the number of preimages for any point on the image of the map is $\bar{n}$, the average acceptance rate is $(|\mathbb{F}_q|/(\bar{n}|\mathcal{G}|)) \cdot \bar{n}/n_{\max} = (|\mathbb{F}_q|/|\mathcal{G}|)/n_{\max}$. □

Use of Map2Point^{-1} for uniformly sampling field elements from $\mathbb{F}_q$. For Map2Point^{-1} from Algorithm 1 the probability of returning $r \neq \perp$ increases proportionally with the number of preimages for a given input. As a result we can use it for transforming a sequence of uniformly sampled group elements $G_l \in \mathcal{G}$ to a sequence of uniformly sampled field elements $r_l \in \mathbb{F}_q$.

Corollary 3.10. *Given a probabilistically invertible map with at most n_{max} preimages according to Definition 3.8, and points G_l uniformly sampled from $\mathcal{G}$, then $r_l \leftarrow$ Map2Point$^{-1}(G_l)$ according to Algorithm 1 outputs results $r_l \neq \perp$ with probability $p \geq (|\mathbb{F}_q|/|\mathcal{G}|)/n_{max}$ and the distribution of outputs $r_l \neq \perp$ will be uniform in $\mathbb{F}_q$.*

Collision probability for map-generated points: Moreover, we can give a bound the collision probability. When sampling two field elements $r_a, r_b \leftarrow_R \mathbb{F}_q$ uniformly from $\mathbb{F}_q$, the probability of a collision $G_a = G_b$, where $G_a \leftarrow \text{Map2Point}(r_a)$ and $G_b \leftarrow \text{Map2Point}(r_b)$, is bounded by $n_{\max}^2/q$.

4 THE CPACE PROTOCOL

As described in Section 1, the CPace protocol [HL19], whose description can be found in Fig. 2, is a SPEKE-like protocol [Jab96] allowing parties to compute a common key via a Diffie-Hellman key exchange with password-dependent generators. Informally, a party $\mathcal{P}$ willing to establish a key with party $\mathcal{P}'$ first computes a generator G from a password pw, a session identifier *sid*, and the party identifiers $\mathcal{P}$ and $\mathcal{P}'$. Next, $\mathcal{P}$ generates an element Y_a from a secret value y_a sampled at random and sends it to $\mathcal{P}'$ together with *sid*. Upon receiving a value Y_b from $\mathcal{P}'$ with respect to same session identifier *sid* and verifying its validity, $\mathcal{P}$ then computes a Diffie-Hellman key K and aborts if K equals the identity element. Finally, it computes the session key as the hash of K together with *sid*, the party identifiers, and the exchanged values Y_a and Y_b.

In order to allow for efficient instantiations over different elliptic curve groups, the CPace description in Fig. 2 contains several specificities: (1) for our analysis the concatenation of party identifiers and exchanged messages uses an ordered concatenation function oc so that messages can be sent in any order and parties do not have to play a specific initiator or responder role (see Appendix C); (2) to avoid directly hashing onto a group or using interactive Map2Point functions, $sid||\text{oc}(\mathcal{P}, \mathcal{P}')||\text{pw}$ is first hashed to the finite field $\mathbb{F}_q$ over which the curve is constructed and then mapped onto the curve via a Map2Point function; (3) the computation of the y and Y values does not assume a particular group and uses generic functions for sampling (SampleScalar) and scalar multiplication (ScalarMult); (4) point verification is performed to protect against trivial attacks against the scheme and merged with scalar multiplication into a ScalarMultVfy function; (5) ephemeral values are erased as soon as they are no longer needed for adaptive security; and (6) transcripts are included in the final key computation to protect against man-in-the-middle attacks.

$\mathcal{P}$		$\mathcal{P}'$
$\mathsf{h} \leftarrow \mathsf{H}_1(sid\|\|\text{oc}(\mathcal{P}, \mathcal{P}')\|\|\text{pw})$		$\mathsf{h}' \leftarrow \mathsf{H}_1(sid\|\|\text{oc}(\mathcal{P}, \mathcal{P}')\|\|\text{pw}')$
$G \leftarrow \text{Map2Point}(\mathsf{h})$		$G' \leftarrow \text{Map2Point}(\mathsf{h}')$
$y_a \leftarrow \text{SampleScalar}()$		$y_b \leftarrow \text{SampleScalar}()$
$Y_a \leftarrow \text{ScalarMult}(G, y_a)$		$Y_b \leftarrow \text{ScalarMult}(G', y_b)$
	$\xleftarrow{(sid, Y_b)}$	
	$\xrightarrow{(sid, Y_a)}$	
$K \leftarrow \text{ScalarMultVfy}(Y_b, y_a)$		$K' \leftarrow \text{ScalarMultVfy}(Y_a, y_b)$
Abort if $K = I_{\mathcal{G}}$		Abort if $K' = I_{\mathcal{G}}$
$ISK \leftarrow \mathsf{H}_2(sid\|\|K\|\|\text{oc}(Y_a, Y_b))$		$ISK' \leftarrow \mathsf{H}_2(sid\|\|K'\|\|\text{oc}(Y_a, Y_b))$
Erase K, pw, y_a, G		Erase K', pw', y_b, G'
Output ISK		Output ISK'

Figure 2: The parallel CPace protocol.

CPace is usually implemented with elliptic curves. In the upcoming section, however, we start with a security analysis of the "basic" CPace protocol on general groups which is shown in Fig. 10. This simplified protocol is parametrized by a security parameter k a group $\mathcal{G}$ and two hash functions H_1 and H_2, both modeled as a random oracle, where H_1 hashes to $\mathcal{G}$ (i.e. without the map).

5 SECURITY OF SIMPLIFIED CPACE

In this Section, as a warm-up, we analyze security of a simplified variant of CPace, which we call CPace$_{\text{base}}$. The main simplification is that we assume the function H_1 to hash onto the group $\mathcal{G}$, such that parties compute generators as $G \leftarrow \mathsf{H}_1(sid||\text{oc}(\mathcal{P}, \mathcal{P}')||\text{pw})$, omitting the map Map2Point. Further, we assume $\mathcal{G}$ to be a multiplicatively written group of prime order p. We further instantiate $\text{ScalarMult}(B, y) := B^y$ as exponentiation, $\text{ScalarMultVfy}(X, y)$ such that it returns the neutral element if X is not in the group and X^y otherwise, and SampleScalar with uniform sampling from $\{1 \ldots p\}$. We postpone dealing with security-related issues introduced by instantiating $\mathcal{G}$ with an elliptic curve, which comes with "imperfect" Map2Point mappings and other artifacts such as X-coordinate-only and cofactor clearing, to a later Section. For clarity, we give a UC execution of CPace$_{\text{base}}$ in Fig. 10 and prove its security properties in the following theorem.

Theorem 5.1 (Security of CPace$_{\text{base}}$). *Let $k, p \in \mathbb{N}$ with p prime and of bit size k. Let $\mathcal{G}$ be a group of order p, and let* $\mathsf{H}_1 : \{0, 1\}^* \rightarrow$

```
The simulator $\mathcal{S}$ (G_2) samples and stores a generator $B \leftarrow \mathcal{G}$.

On $(\mathsf{NewSession}, sid, P_i, P_j)$ from $\mathcal{F}_{\mathsf{lePAKE}}$:
(G_4) sample $z_i \leftarrow_{\mathrm{R}} \mathbb{Z}_p$, set $Y_i \leftarrow B^{z_i}$, store $(P_i, z_i, Y_i, \bot)$
(G_4) send $Y_i$ to $\mathcal{A}$ intended to $P_j$

On $Z^*$ from $\mathcal{A}$ as msg to $(sid, P_i)$:
(G_4) if $Z^*$ is adversarially generated and $Z^* \in \mathcal{G} \setminus I_{\mathcal{G}}$,
          then send $(\mathsf{RegisterTest}, sid, P_i)$ to $\mathcal{F}_{\mathsf{lePAKE}}$

Upon $P_i$ receiving $Y_j \in \mathcal{G}$ from $P_j$:
(G_4) retrieve record $(P_i, z_i, Y_i, *)$
(G_4) if $\exists$ records (G_5) $(\mathsf{H}_1, \mathrm{oc}(P_i, P_j), \mathrm{pw}, r, r^{-1}, G)$, $(\mathsf{H}_2, K||\mathrm{oc}(Y_i, Y_j), ISK)$ such that $K = Y_j^{z_i r^{-1}}$
(G_5)       store $(\mathsf{guess}, G, Y_j)$, abort if $\exists$ record $(\mathsf{guess}, G', Y_j)$ with $G \neq G'$ and (G_4) send $(\mathsf{TestPwd}, sid, P_i, \mathrm{pw})$ to $\mathcal{F}_{\mathsf{lePAKE}}$
(G_4)       send $(\mathsf{NewKey}, sid, P_i, ISK)$ to $\mathcal{F}_{\mathsf{lePAKE}}$ and store $(P_i, z_i, Y_i, ISK)$
(G_4) else sample a fresh random $ISK'$ and send $(\mathsf{NewKey}, sid, P_i, ISK')$ to $\mathcal{F}_{\mathsf{lePAKE}}$ //$\mathcal{F}_{\mathsf{lePAKE}}$ will discard $ISK'$

On $\mathsf{H}_1(sid||\mathcal{P}||\mathcal{P}'||\mathrm{pw})$ from $\mathcal{A}$:
(G_2) if this is the first such query then
(G_2)       create one by sampling $r \leftarrow_{\mathrm{R}} \mathbb{F}_p \setminus \{0\}$
(G_3)       abort if there exists a record $(\mathsf{H}_1, *, *, r, *, *)$
(G_2)       store $(\mathsf{H}_1, \mathcal{P}||\mathcal{P}', \mathrm{pw}, r, r^{-1}, B^r)$ and set $h \leftarrow B^r$
(G_2) else retrieve record $(\mathsf{H}_1, \mathcal{P}||\mathcal{P}', \mathrm{pw}, *, *, h)$
(G_2) reply with $h$

On receiving $(\mathsf{AdaptiveCorruption}, sid)$ from $\mathcal{A}$ as msg to $P_i$:
send $\mathsf{AdaptiveCorruption}, sid, P_i)$ to $\mathcal{F}_{\mathsf{lePAKE}}$
retrieve record $(sid, \mathrm{pw})$
(G_4) if a message $Y_i := B^{z_i}$ was already sent to $P_j$, then
(G_4)       if a record $(\mathsf{H}_1, *, \mathrm{pw}, r, r^{-1}, *)$ exists then set $y_i \leftarrow z_i r^{-1}$
(G_4)       else store $(\mathsf{H}_1, *, \mathrm{pw}, r', r'^{-1}, B^{r'})$, $r' \leftarrow_{\mathrm{R}} \mathbb{Z}_p$ and set $y_i \leftarrow z_i r'^{-1}$
(G_4)       send $(\mathrm{pw}, y_i)$ to $\mathcal{A}$

On $\mathsf{H}_2(sid||K||Y_i||Y_j)$ from $\mathcal{A}$:
(G_4) if this is the first such query then
(G_7)       if $\exists$ records $(P_i, z_i, Y_i, *)$, $(P_j, z_j, Y_j, *)$, $(\mathsf{H}_1, \mathrm{oc}(P_i, P_j), r, r^{-1}, *)$ such that $K^r = B^{z_i z_j}$ then abort
(G_4)       if $\nexists$ records $(P_i, *, Y_i, *)$ or $(P_j, *, Y_j, *)$, or if $Y_a||Y_b \neq \mathrm{oc}(Y_a, Y_b)$, then sample $A \leftarrow_{\mathrm{R}} \{0,1\}^{2k}$
(G_4)       if $\exists$ records $(P_i, z_i, Y_i, ISK)$ and (G_5) $(\mathsf{H}_1, \mathrm{oc}(P_i, P_j), \mathrm{pw}, r, r^{-1}, G)$ such that $K = Y_j^{z_i r^{-1}}$, (G_5) record $(\mathsf{guess}, G, Y_j)$, abort if
                $\exists$ record $(\mathsf{guess}, G', Y_j)$ with $G \neq G'$. Send $(\mathsf{LateTestPwd}, sid, P_i, \mathrm{pw})$ to $\mathcal{F}_{\mathsf{lePAKE}}$. Upon answer $\hat{K}$ set $A \leftarrow \hat{K}$
(G_4)       if $\exists$ records $(P_j, z_j, Y_j, ISK)$ with $ISK \neq \bot$ and (G_5) $(\mathsf{H}_1, \mathrm{oc}(P_i, P_j), \mathrm{pw}, r, r^{-1}, G)$ such that $K = Y_i^{z_j r^{-1}}$, (G_5) store $(\mathsf{guess}, G, Y_i)$,
                abort if $\exists$ record $(\mathsf{guess}, G', Y_i)$ with $G \neq G'$. Send $(\mathsf{LateTestPwd}, sid, P_j, \mathrm{pw})$ to $\mathcal{F}_{\mathsf{lePAKE}}$. Upon answer $\hat{K}$ set $A \leftarrow \hat{K}$
(G_4)       if no matching $\mathsf{H}_1$ records are found set $A \leftarrow_{\mathrm{R}} \{0,1\}^{2k}$
(G_4)       finally, store $(\mathsf{H}_2, K||Y_i||Y_j, A)$ and reply with $A$
(G_4) else retrieve record $(\mathsf{H}_2, K||Y_i||Y_j, A)$ and reply with $A$
```

Figure 3: Simulator for CPace$_{\text{base}}$, with game numbers to indicate which game introduces a particular line of code (cf. full proof of Theorem 5.1 in Appendix A. For brevity we omit session identifiers *sid* from all records stored by the simulator.

$\mathcal{G}, \mathsf{H}_2 : \{0,1\} \to \{0,1\}^k$ be two hash functions. If the sCDH and sSDH problems are hard in $\mathcal{G}$, then protocol CPace$_{\text{base}}$ depicted in Fig. 10 UC-emulates $\mathcal{F}_{\mathsf{lePAKE}}$ in the random oracle model with respect to adaptive corruptions and both hash functions modeled as random oracles. More detailed, it holds that

$$|Pr[Real_{\mathcal{Z}}(CPace_{\text{base}}, \mathcal{A})] - Pr[Ideal_{\mathcal{Z}}(\mathcal{F}_{\mathsf{lePAKE}}, \mathcal{S})| \leq l_{\mathsf{H}_1}^2/p + 2l_{\mathsf{H}_1}^2 \mathbf{Adv}^{sSDH} + \mathbf{Adv}^{sCDH}$$

where l_{H_1} denotes the number of H_1 queries made by the adversary $\mathcal{A}$ and simulator $\mathcal{S}$ is depicted in Fig. 3.

Proof Sketch. The main idea of the simulation is to fix a secret generator $B \in \mathcal{G}$ and carry out the simulation with respect to B. Messages of honest parties are simulated as B^z for a fresh exponent z. Queries $\mathsf{H}_1(sid||\mathrm{oc}(\mathcal{P}, \mathcal{P}')||\mathrm{pw})$ are answered with B^r for a freshly chosen "trapdoor" r. The simulator might learn an honest party's password via adaptive corruption or via an adversarial password guess. The simulator can now adjust the simulation in retrospective to let the honest party use $B^r = \mathsf{H}_1(sid||\mathrm{oc}(\mathcal{P}, \mathcal{P}')||\mathrm{pw})$ by claiming the party's secret exponent to be zr^{-1}. This already concludes simulation of honest parties without passwords. Adversarial password guesses can be read from $\mathcal{A}$ injecting X (or, similarly, Y) and then querying $\mathsf{H}_2(sid||K||X||Y)$ with K being a correctly computed key w.r.t some generator B^r provided by the simulation. $\mathcal{S}$ can now read the guessed password from the H_1 list, and submit it as password guess to $\mathcal{F}_{\mathsf{lePAKE}}$. In case of success, the simulator sets the key of the honest party to $\mathsf{H}_2(sid||K||X||Y)$.

The simulation is complicated by the order of honest parties' outputs (which are generated upon receipt of the single message) and the adversary's computation of final session keys via H_2 queries. If the key is generated by $\mathcal{F}_{\mathsf{lePAKE}}$ before $\mathcal{A}$ computes it via H_2 (which constitutes a password guess as detailed above), then $\mathcal{S}$ needs to invoke the LateTestPwd query of $\mathcal{F}_{\mathsf{lePAKE}}$ instead of TestPwd. In case of a correct guess, this lets $\mathcal{S}$ learn the output key of the honest party, which $\mathcal{S}$ can then program into the corresponding H_2 query.

Finally, the simulation can fail in some cases. Firstly, $\mathcal{S}$ might find more than one password guesses against an honest party with simulated message X. In this case simulation cannot continue since $\mathcal{F}_{\mathsf{lePAKE}}$ allows for only one password guess per party. In this case, however, $\mathcal{A}$ would provide (B^r, X, Y, K),$(B^{r'}, X, Y, K')$ which are two CDH tuples for passwords pw, pw′ with $B^r \leftarrow \mathsf{H}_1(\mathsf{pw})$, $\mathsf{pw}' \leftarrow \mathsf{H}_1(\mathsf{pw}')$ (omitting session and party identifiers for brevity). Provided that the simultaneous strong CDH assumption (sSDH, cf. Definition 3.2) holds, this cannot happen. Here, the "strong" property, providing a type of DDH oracle, is required to help $\mathcal{S}$ identify CDH tuples among all queries to H_2. A second case of simulation failure occurs when $\mathcal{A}$ wants to compute a key of an uncorrupted session via a H_2 query. Since $\mathcal{S}$ does not know such keys, it would have to abort. Using a similar strategy as above, pseudorandomness of keys can be shown to hold under the strong CDH assumption, and thus the probability of $\mathcal{A}$ issuing such a H_2 query is negligible. The full proof can be found in Appendix A. □

5.1 Embedding CDH experiment libraries into the simulator

In this section, we discuss an alternative approach to carrying out reductions to cryptographic assumptions in the case of CPace/CDH. Both assumptions required by $\mathsf{CPace}_{\mathsf{base}}$, sCDH and sSDH, allow for an *efficient implementation* in the following sense: the secret exponents that are sampled by the experiment code (often also called the *challenger*) for generating challenges are sufficient for answering the restricted DDH queries allowed by both assumptions. An example for an assumption that does not allow for such efficient instantiation is, e.g., gap CDH. In gap-CDH, the adversary is provided with a "full" DDH oracle that he can query on arbitrary elements, of which the experiment might not know an exponent for.

Due to this property, we can integrate implementations of the CDH experiments in the simulator's code. The simulator will thus implement the DDH oracles on its own, and abort if at any time an oracle query solves the underlying assumption. We chose to integrate experiments as libraries (written as objects in python-style notation) into the simulator's code. This eases not only presentation but will also become useful when analyzing variants of CPace that require slightly different assumptions.

The corresponding result for $\mathsf{CPace}_{\mathsf{base}}$ is shown in Fig. 4. The sCDH class in Fig. 4 produces a challenge consisting of two uniformly drawn group elements $Y_1 \leftarrow B^{y_1}, Y_2 \leftarrow B^{y_2}$. The limited DDH oracle provided by the sCDH assumption can only receive inputs w.r.t one of these elements, and thus it can be implemented efficiently using secret exponents y_1, y_2. If a correct CDH solution $B, Y_1, Y_2, B^{y_1 \cdot y_2}$ is provided, the library aborts. The simulator from Fig. 3 is adapted to call the libraries. As an example, honest parties' messages are simulated by calling the challenge sampling procedure exp.sampleY().

Proving indistinguishability. With this simulation approach, a proof consists in demonstrating that ideal and real world executions are indistinguishable except for events in which the experiment libraries abort because a challenge was correctly answered. Compared to our proof of Theorem 5.1, the indistinguishability argument becomes simpler because the reduction strategies to both CDH-type assumptions are already embedded in the corresponding assumption experiment libraries. Losses such as the factor of $2l^2_{\mathsf{H}_1}\,\mathbf{Adv}^{\mathrm{ssCDH}}$ in the reduction to sSDH in game $\mathbf{G}_5$ translate to libraries producing more than one challenge per simulation run, as is the case for the sSDH experiment from Fig. 4. Altogether, the simulation with integrated CDH experiment libraries is an alternative approach of proving Theorem 5.1, as we formalize in the following.

Theorem 5.2 (Alternative simulation for Theorem 5.1)**.** *The simulator depicted in Fig. 4 is a witness for the UC emulation statement in Theorem 5.1*

Proof sketch. The output distribution of the simulator $\mathcal{S}$ from Fig. 4 is indistinguishable from the one of the simulator from Fig. 3 as it is obtained from internal restructuring. $\mathcal{S}$ aborts if either the sSDH or the sCDH experiment class aborts, which occurs iff a correct solution has been provided to the experiment implementation or a H_1 collision is observed. These cases coincide with the abort cases in the proof of Theorem 5.1. As the sSDH object outputs $2l^2_{\mathsf{H}_1}$ different challenges and as it is sufficient for $\mathcal{Z}$ to provide a solution to one of these challenges for distinguishing both worlds, the advantage for solving the sSDH problem needs to be multiplied by this factor, thus reproducing the bounds from Theorem 5.1. □

Advantages of embedding libraries in the simulation. To clarify, the approach presented in this section does *not* allow to prove stronger security statements. As demonstrated above, it is merely an alternative way of presenting security proofs in the UC framework or other simulation-based frameworks, and it works whenever the underlying cryptographic assumptions are efficiently implementable. However, we believe that the approach has its merits especially in the following dimensions.

- **Modular security analysis.** Slight modifications in the protocol might require to change the cryptographic assumption. As long as the public interface does not change, our approach allows to switch between assumptions by simply calling a different library. Cryptographers then need to only analyze this "local" change in the simulation, which prevents them from re-doing the whole indistinguishability argument.
- **Presentation of reduction strategies.** In normal game-based indistinguishability arguments [Sho04], reductions to cryptographic assumptions are hidden within side-long proofs. With our approach, the reduction strategy is depicted in clear code as part of the simulator's code. This makes checking of proofs easier not only for readers but also might make simulation-based proofs more accessible to automated verification.

In this paper, our motivation is the first dimension. In the upcoming section, the library-based approach will turn out to be extremely useful to analyze the various variants of CPace that stem from (efficiency-wise) optimized implementations on different elliptic curves.

6 ANALYSIS OF REAL-WORLD CPACE

The currently most efficient way to run CPace is over elliptic curves. Therefore, from this point onwards, we consider $\mathcal{G}$ to be an elliptic curve constructed over field $\mathbb{F}_q$. From a historical perspective, both CPace research and implementation first focused on prime order

```
# using python-style notation with self pointer $s$ for accessing members in methods and _init_ constructor
def class sCDH:
    def _init_($s, G$): { $s.B \leftarrow G$; $s.i \leftarrow 0$;  $s$.state← fresh; }
    def sampleY(s): { if $s.i \leq 2$: {$s.i+=1$; sample $s.y_i \leftarrow_R \mathbb{F}_p \setminus 0$; return $(s.B)^{s.y_i}$; } }
    def corrupt(s,$X$): { for $1 \leq m \leq s.i$): {if $(X = (s.B)^{s.y_m})$: $x \leftarrow s.y_m$; $s$.state← corrupt; return x; } }
    def DDH($s, B, Y, X, K$):
        if ($\{Y, X\}$=$\{s.Y_1, s.Y_2\}$) and ($s$.state= fresh) and $(K = (s.B)^{s.y_1 \cdot s.y_2})$ : abort("sCDH($B, Y_1, Y_2$) solved");
        for $1 \leq m \leq s.i$: { if $(Y = (s.B)^{s.y_m})$: return $(K = X^{s.y_m})$; } # compute DDH oracle reply from secret exponent of $Y$
     def isValid($X$): return $(X \in \mathcal{G} \setminus I_{\mathcal{G}})$
def class sSDH: # using python-style notation [ ] for list containers
    def _init_($s$,sCdhExp): { sample $s.B \leftarrow_R \mathcal{G}$; $s.scdh$ = sCdhExp($s.B$); $s$.records =[ ]; $s$.guess = "yet no guess"; }
    def sampleY($s$): { return ($s.scdh$).sampleY(); }
    def isValid($X$): return ($s.scdh$).isValid($X$);
    def sampleH1($s$): { sample $r \leftarrow_R \mathbb{F}_p \setminus 0$; if: $r$ in $s$.records abort("H1 collision"); else: {$s$.records.append(($r$, $(s.B)^r$)); return $(s.B)^r$; }}
    def corrupt($s, G, Y$): { if there is $(r, G)$ in $s$.records: return ($s.scdh$).corrupt($Y^{1/r}$); }
    def DDH($s$,$G, Y, X, K$):
        if there is $(r, G)$ in $s$.records:
            match ← ($s.scdh$).DDH($s.B, Y, X, K^{1/r}$);
            if match and (guess = "yet no guess"): (guess.$G$,guess.$X$)← $(G, X)$;
            elif match and (guess.$X = X$) and (guess.$G \neq G$): abort("sSDH problem ($Y, G$,guess.$G$) solved");
            return match;
```

```
On first invokation $\mathcal{S}$ creates an experiment class instance exp= sSDH(sCDH);
```

```
On (NewSession, $sid, P_i, P_j$) from $\mathcal{F}_{\text{lePAKE}}$:
($G_4$) Set $Y_i \leftarrow$ exp.sampleY(), store $(P_i, P_j, Y_i, \perp)$
($G_4$) send $Y_i$ to $\mathcal{A}$ intended to $P_j$
```

```
On $Z^*$ from $\mathcal{A}$ as msg to $(sid, P_i)$:
($G_4$) if $Z^*$ is adversarially generated and exp.isValid($Z^*$)
($G_4$)             send (RegisterTest, $sid, P_i$) to $\mathcal{F}_{\text{lePAKE}}$
```

```
Upon $P_i$ receiving $Y_j$ from $P_j$: retrieve record $(P_i, *, z_i, Y_i, *)$
($G_1$) if not exp.isValid($Y_j$): return;
($G_4$) if ∃ records ($H_1$, pw,oc$(P_i, P_j), h$), ($H_2, K||$(oc$(Y_i, Y_j), ISK$) such that exp.DDH($h, Y_i, Y_j, K$) = 1:
($G_4$)       send (TestPwd, $sid, P_i$, pw) to $\mathcal{F}_{\text{lePAKE}}$
($G_4$)       send (NewKey, $sid, P_i, ISK$) to $\mathcal{F}_{\text{lePAKE}}$ and store $(P_i, P_j, Y_i, ISK)$
($G_4$) else sample a fresh random $ISK'$ and send (NewKey, $sid, P_i, ISK'$) to $\mathcal{F}_{\text{lePAKE}}$ # $\mathcal{F}_{\text{lePAKE}}$ will discard $ISK'$
```

```
On $H_1(sid||P_i||P_j||$pw)) from $\mathcal{A}$:
($G_2$) if this is the first such query then
($G_2$)      $h \leftarrow$ exp.sampleH1()
($G_2$)      store ($H_1$, pw, $P_i||P_j, h$)
($G_2$) lookup ($H_1$, pw, $P_i||P_j, h$) and reply with $h$
```

```
On receiving (AdaptiveCorruption, $sid$) from $\mathcal{A}$ as msg to $P_i$:
Lookup $(P_i, P_j, Y_i, *)$; send (AdaptiveCorruption, $sid, P_i$) to $\mathcal{F}_{\text{lePAKE}}$, obtain ($sid$, pw)
($G_4$) if a message $Y_i$ was already sent to $P_j$, then
($G_4$)       query $H_1$ for ($sid||$pw, oc$(P_i, P_j)$) and retrieve record ($H_1$, pw, oc$(P_i, P_j), h$)
($G_4$)       send (pw,exp.corrupt($h, Y_i$))
```

```
On $H_2(sid||K||Y_i||Y_j)$ from $\mathcal{A}$:
($G_4$) Lookup ($H_2$, $sid||K||Y_i||Y_j, h$) and send $h$ if it exists; else if this is the first such query:
($G_4$)      if there are no records $(P_i, P_j, Y_i, *)$ or $(P_j, P_i, Y_j, *)$, or if $Y_a||Y_b \neq$ oc$(Y_a, Y_b)$: sample $A \leftarrow \{0,1\}^{2k}$;
($G_4$)      if ∃ records $(P_i, P_j, Y_i, ISK)$ with $ISK \neq \perp$ and ($H_1$, pw, oc$(P_i, P_j), h$) such that exp.DDH($h, Y_i, Y_j, K$) = 1:
($G_5$)            send (LateTestPwd, $sid, P_i$, pw) to $\mathcal{F}_{\text{lePAKE}}$. Upon answer $\hat{K}$ set $A \leftarrow \hat{K}$
($G_4$)      if ∃ records $(P_j, P_i, Y_j, ISK)$ with $ISK \neq \perp$ and ($H_1$, pw, oc$(P_i, P_j), h$) such that exp.DDH($h, Y_j, Y_i, K$) = 1:
($G_5$)            send (LateTestPwd, $sid, P_j$, pw) to $\mathcal{F}_{\text{lePAKE}}$. Upon answer $\hat{K}$ set $A \leftarrow \hat{K}$
($G_4$)      if no matching $H_1$ records are found set $A \leftarrow \{0,1\}^{2k}$
($G_4$)      finally, store ($H_2$, $sid||K||Y_i||Y_j, A$) and reply with $A$
```

Figure 4: Simulator for "basic" parallel CPace embedding CDH challenges generated by libraries

curves, such as the NIST-P-256 curve [DSS13]. Subsequently significantly improved performance was shown on Montgomery- and (twisted-)Edwards curves, notably Curve25519 and Ed448 curves [Ber06, Ham15b], which both have a small cofactor c in their group order $c \cdot p$. These approaches consider also implementation pitfalls, e.g., by designing the curve such that there are no incentives for implementers to use insecure speed-ups. Thirdly, recently ideal group abstractions have been presented in order to avoid the complexity of small cofactors in the group order [Ham15a, dVGT+20], while maintaining all advantages of curves with cofactor.

For smooth integration into each of these different curve ecosystems, CPace needs to be instantiated slightly different regarding, e.g., computation of the DH generator, group size, multiplication and sampling algorithms. In this section, we analyze how such differences impact security. Using our modular approach with assumption libraries called by a simulator, we are able to present security in terms of differences from our basic CPace analysis in Section 5 in a concise way.

6.1 CPace without Hashing to the Group

In this subsection we analyze the CPace protocol as depicted in Fig. 2 and where $\mathcal{G}$ is an elliptic curve constructed over some field $\mathbb{F}_q$. The only difference to simplified CPace analyzed in the previous section is how parties compute the generators: now the function H_1 hashes onto the field $\mathbb{F}_q$, and generators are computed as $G \leftarrow \mathsf{Map2Point}(\mathsf{H}_1(sid||\mathsf{oc}(P_i, P_j)||\mathsf{pw})$ for a map $\mathsf{Map2Point} : \mathbb{F}_q \rightarrow \mathcal{G}$. This way, the H_1 outputs can be considered to form an alternative encoding of group elements, where Map2Point decodes to the group. ScalarMult, ScalarMultVfy and SampleScalar are as in Section 5.

Security analysis. Security analysis is complicated by this change in essentially two ways: first, the possibly non-uniform distribution of Map2Point induces non-uniformity of DH generators computed by the parties. Second, embedding of trapdoors no longer works by simply programming elements with known exponents into H_1. Instead, the proof will exploit that Map2Point is probabilistically invertible, such that preimages of generators with known exponents can be programmed into H_1 instead. Consequently, security of CPace will be based on the $\mathcal{D}_\mathcal{G} -$sSDH problem Definition 3.3 instead of the sSDH problem, where the distribution $\mathcal{D}_\mathcal{G}$ corresponds to the distribution of group elements $\mathsf{Map2Point}(h_i)$ obtained for uniformly sampled field elements $h_i \leftarrow_{\mathrm{R}} \mathbb{F}_q$. All these changes can be captured by replacing library sSDH with a new library for $\mathcal{D}_\mathcal{G} -$ sSDH, as we demonstrate below.

Theorem 6.1 (Security of CPace with Map2Point)**.** *Let $k, p, q \in \mathbb{N}$ with p prime and of bit size k. Let $\mathcal{G}$ an elliptic curve of order p over field $\mathbb{F}_q$. Let $\mathsf{H}_1 : \{0,1\}^* \rightarrow \mathbb{F}_q$, $\mathsf{H}_2 : \{0,1\}^* \rightarrow \{0,1\}^k$ be two hash functions and Map2Point : $\mathbb{F}_q \rightarrow \mathcal{G}$ probabilistically invertible with bound n_{max}. Let $\mathcal{D}_\mathcal{G}$ denote the distribution on $\mathcal{G}$ induced by Map2Point. If the sCDH and $\mathcal{D}_\mathcal{G}$-sSDH problems are hard in $\mathcal{G}$, then the CPace protocol depicted in Fig. 2 UC-emulates $\mathcal{F}_{\mathsf{lePAKE}}$ in the random oracle model with respect to adaptive corruptions and both hash functions modeled as random oracles. More detailed, it holds that*

$$|Pr[Real(\mathcal{Z}, \mathcal{A}, CPace)] - Pr[Ideal(\mathcal{F}_{\mathsf{lePAKE}}, \mathcal{S})|$$
$$\leq (l_{\mathsf{H}_1})^2/p + (n_{max} \cdot l_{\mathsf{H}_1})^2/q + 2l_{\mathsf{H}_1}^2 \mathbf{Adv}^{D_\mathcal{G}-sSDH} + \mathbf{Adv}^{sCDH}$$

where $\mathrm{l}_{\mathsf{H}_1}$ denotes the number of H_1 queries made by the adversary $\mathcal{A}$ and simulator $\mathcal{S}$ is as in Fig. 4 but using the object distExp (cf. Fig. 5) instead of the object sSdhExp.

Proof Sketch. We adjust the simulator for "basic" CPace from Fig. 4 as follows. First, we embed the reduction strategy from Theorem 3.6 into an experiment library that converts sSDH challenges into $\mathcal{D}_\mathcal{G} -$ sSDH challenges and obtain the class $\mathcal{D}_\mathcal{G}$_sSDH depicted in Fig. 5. The class $\mathcal{D}_\mathcal{G}$_sSDH uses the Map2PointPreImages function (passed as a constructor parameter) for implementing the $\mathsf{Map2Point}^{-1}$ as defined in Algorithm 1 and an instance of the sSDH class implementing a sSDH experiment and an instance of the sSDH class that is assigned to a member variable. Note that $\mathsf{Map2Point}^{-1}$ is a rejection sampler for $\mathcal{D}_\mathcal{G}$ by Lemma 3.9 and, correspondingly, Theorem 3.6 applies. Each time the main body of the simulator from Fig. 4 makes calls to its exp object, the corresponding method of the new $\mathcal{D}_\mathcal{G}$_sSDH object will be executed, which itself translates the queries into calls to the sSDH object that was passed as constructor parameter.

Importantly, $\mathcal{D}_\mathcal{G}$_sSDH provides the same public API as the sSDH class with the distinction that sampling for H_1 returns results from $\mathbb{F}_q$ instead of $\mathcal{G}$. Moreover $\mathcal{D}_\mathcal{G}$_sSDH aborts if the code of its sSDH object aborts and upon H_1 collisions.

We explain now how the indistinguishability argument of Theorem 5.1 needs to be adjusted in order to work for Theorem 6.1 and this new simulator. The first difference applies in game $\mathbf{G}_2$, where we must make sure that the distribution of points provided by the $\mathcal{D}_\mathcal{G}$_sSDH object is uniform in $\mathbb{F}_q$ as was in the previous game. This is the case due to Corollary 3.10. In game $\mathbf{G}_3$ no change is needed except for adjusting the collision probability following the derivation from Section 3.3 as which is now bound by $(n_{\max} \cdot l_{\mathsf{H}_1})^2/q$ in addition to the previous $\mathrm{l}_{\mathsf{H}_1}^2/p$ probability. Apart of these modification the proof applies without further changes.

□

Actual implementations of Map2Point. The property of *probabilistic invertibility* is fulfilled for a wide variety of mapping algorithms, specifically those currently suggested in [FHSS+19], i.e. Elligator2 [BHKL13], simplified SWU [FHSS+19] and the Shallue-van de Woestijne method (SvdW) [SvdW06]. The most generic of these algorithm, SvdW, works for all elliptic curves, while the simplified SWU and Elligator2 algorithms allow for more efficient implementations given that the curve fulfills some constraints.

All these mappings have a fixed bound $n_{\max}$ regarding the number of pre-images and come with a PPT algorithm for calculating all preimages. For instance, Elligator2 [BHKL13] comes with a maximum $n_{\max} = 2$ of two pre-images per point and $n_{\max} \leq 4$ for the simplified SWU and SvdW algorithms [FHSS+19]. For all these algorithms, the most complex substep for determining all pre-images is the calculation of a small pre-determined number of square roots and inversions in $\mathbb{F}_q$ which could be easily implemented in polynomial time with less computational complexity than one exponentiation operation.

Map-twice-and-add constructions. Some mapping constructions aim at producing more uniform distributions using a map-twice-and-add approach, such as presented in [FHSS+19] and also adopted by the ristretto25519 and decaf ecosystems [dVGT+20, Ham15a]. Variant $\mathrm{CPace}_{\mathsf{mapTwice}}$ computes generators as $G \leftarrow \mathsf{Map2Point}(h) \cdot \mathsf{Map2Point}(h')$, where $(h, h') \leftarrow \mathsf{H}_1(sid||\mathsf{oc}(\mathcal{P}, \mathcal{P}')||\mathsf{pw})$. Following our modular simulation approach, we give the corresponding experiment library mapTwice_sSDH in Fig. 11. The library produces two elements from $\mathbb{F}_q$ instead of a single one as before. Again we make use of the property of probabilistic invertibility which guarantees that the two field elements output by the sampleH1() function are uniformly distributed in $\mathbb{F}_q$. Here the collision bound $\leq l_{\mathsf{H}_1}^2/p$ is

```
# using python-style notation with self pointer $s$
def class $\mathcal{D}_\mathcal{G}$_sSDH:
    def _init_($s$, Map2PointPreImages, $n_{\max}$, sSDHExp):
        $s.sSDH$ = sSDHExp; $s.records$ = [];
        $s.nmax$ = $n_{\max}$; $s.preim$ = Map2PointPreImages;
    def sampleY($s$): return $(s.sSDH)$.sampleY();
    def isValid($X$): return $(s.sSDH)$.isValid($(X)$);
    def sampleH1($s$):
        $G \leftarrow (s.sSDH)$.sampleH1();
        while (1):
            sample $r \leftarrow_R \mathbb{F}_p$; preimageList = $(s.preim)(G^r)$;
            sample $m \leftarrow_R \{0 \dots (s.n_{\max} - 1)\}$;
            if len(preimageList) > $m$:
                if $r = 0$: abort("Sampled neutral element.");
                $h \leftarrow$preimageList$[m]$;
                if $h$ in $s.records$ abort("H1 collision");
                $s.records$.append($r, G^r, h$);
                return $h$;
    def corrupt($s, h, Y$):
        if there is $(r, G, h)$ in $s$.records:
            return $(s.sSDH)$.corrupt($G, Y^{1/r}$);
    def DDH($s$,$h, Y, X, K$):
        if there is $(r, G, h)$ in $s$.records:
            return $(s.sSDH)$.DDH$(G, Y, X, K^{1/r})$;
# Chaining the experiments for CPace on prime order curve,
# full (x,y) coordinate, single map execution
sSdhExp = sSDH(sCDH);
distExp = $\mathcal{D}_\mathcal{G}$_sSDH(Map2PointPreImages, $n_{\max}$,sSdhExp);
```

Figure 5: Experiment class definition $\mathcal{D}_\mathcal{G}$-sSDH using single executions of Map2Point, where H_1 hashes to $\mathbb{F}_q$.

maintained at the expense of doubling the computational complexity of the map [Ham20, BCI+10b, FHSS+19, FT12]. Security holds under the same assumption set as for "map2point-once" (Theorem 6.1), specifically Map2Point needs to be probabilistically invertible such that the rejection samplers in Fig. 11 can be implemented and the simulated H_1 outputs become uniform in $\mathbb{F}_q \times \mathbb{F}_q$.

6.2 Considering curves with small co-factor

In this subsection, we now additionally consider that the elliptic curve group can be of order $c \cdot p$ where Map2Point maps to the full curve, denoted $\mathcal{G}_{c \cdot p}$, but security guarantees could only be given for the subgroup of order p, denoted $\mathcal{G}_p$. Consequently, CPace$_{\text{co}}$ on curves with co-factor c requires all secret exponents to be multiples of c. Hence, CPace$_{\text{co}}$ is CPace depicted in Fig. 2 where ScalarMult : $(X, y) \mapsto (X^{c \cdot y})$ and ScalarMultVfy is defined accordingly, with the addition that the latter returns I in case of inputs not on the curve $\mathcal{G}_{c \cdot p}$.

Theorem 6.2 (Security of CPace$_{\text{co}}$). *Let $k, p, q, c \in \mathbb{N}$, p, c co-prime with p prime. Let $\mathcal{G}_{c \cdot p}$ an elliptic curve of order $p \cdot c$ over field $\mathbb{F}_q$ and $\mathcal{G}_p \subset \mathcal{G}_{p \cdot c}$ a subgroup of order p . Let $CC_c : (G) \mapsto ((G^c)^{1/c \mod p})$ be a cofactor clearing function for c, $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{F}_q$, $\mathsf{H}_2 : \{0,1\}^* \to \{0,1\}^k$ be two hash functions and Map2Point : $\mathbb{F}_q \to \mathcal{G}_{c \cdot p}$ probabilistically invertible with bound n_{max}. Let $\mathcal{D}_\mathcal{G}$ denote the distribution on $\mathcal{G}_p$ induced by the chained function ($CC_c \circ$ Map2Point). If the sCDH and $\mathcal{D}_\mathcal{G}$-sSDH problems are hard in $\mathcal{G}_{p \cdot c}$, then CPace$_{\text{co}}$ UC-emulates $\mathcal{F}_{\text{lePAKE}}$ in the random oracle model with respect to adaptive corruptions and both hash functions modeled as random oracles. More detailed, it holds that*

$$|Pr[Real(\mathcal{Z}, \mathcal{A}, CPace_{\text{co}})] - Pr[Ideal(\mathcal{F}_{\text{lePAKE}}, \mathcal{S})|$$
$$\leq (n_{max} \cdot c \cdot l_{\mathsf{H}_1})^2/q + 2l^2_{\mathsf{H}_1} \mathbf{Adv}^{D_\mathcal{G}-sSDH} + \mathbf{Adv}^{sCDH}$$

where l_{H_1} denotes the number of H_1 queries made by the adversary $\mathcal{A}$ and simulator $\mathcal{S}$ is as in Fig. 4 but using class ccDistExp (cf. Fig. 6) instead of class sSDH.

```
# using python-style notation with self pointer $s$
def class cofactorClearer:
    "interfaces $\mathcal{S}$ to a prime-prder experiment class"
    def _init_($s, c, p$, primeOrderExpInstance ,$p_{twist}$):
        $s.c$ = c; $s$.i= $s.c \cdot$integer$(1/(s.c^2) \mod p)$;
        $s$.it= $s.c \cdot$ integer$(1/(s.c^2) \mod p_{twist})$;
        $s$.exp = primeOrderExpInstance;
    def sampleY($s$): return $((s.\text{exp}).\text{sampleY}())^{s.c}$;
    def isValid($X$): return $(s.\text{exp})$.isValid$(X^{s.i})$
    def sampleH1($s$): return $(s$.exp$)$.sampleH1();
    def corrupt($s, h, Y$): { return $(s$.exp$)$.corrupt($h, Y^{s.\text{i}}$); }
    def DDH($s$,$G, Y, X, K$):
        if $X \in \mathcal{G}$ return $(s.\text{exp})$.DDH$(G, Y^{s.\text{i}}, X^{s.\text{i}}, K^{s.\text{i}^2})$
        if $X$ on twist return $(s.\text{exp})$.DDH$(G, Y^{s.\text{it}}, X^{s.\text{it}}, K^{s.\text{it}^2})$

sSdhExp = sSDH(sCDH);
ccExp = cofactorClearer(sSdhExp);
ccDistExp = $\mathcal{D}_\mathcal{G}$_sSDH(Map2PointPreImages, $n_{\max}$,ccExp);
```

Figure 6: Cofactor-clearer class definition use for elliptic curves of order $p \cdot c$ with a quadratic twist of order p_{twist}. Note that the inverses $s.i$ and $s.it$ are constructed such that they are multiples of c.

Proof Sketch. The group $\mathcal{G}_{c \cdot p}$ has a point B_1 of order c with $B_i^c = I$ where I denotes the identity element in $\mathcal{G}_{p \cdot c}$, i.e., there are c low-order points $B_1^i, i \in \{1 \dots c\}$. For any point $Y \in \mathcal{G}_{p \cdot c}$ we can consider the points $Y_i = Y + B_i$ as alternative ambiguous representations of the point $CC_c(Y)$. For any input point $Y \in \mathcal{G}_{p \cdot c}$, all these c alternative representations can be easily calculated using group operations and B_i. For any of these c alternative representations of Y at most $n_{\max}$ preimages will be returned by Map2PointPreImages$_{\mathcal{G}_{c\dot{p}}}$ since Map2Point is probabilistically invertible on $\mathcal{G}_{p \cdot c}$. If up to $n_{\max}$ preimages exist per point on the full curve, Map2Point is probabilistically invertible also on $\mathcal{G}_p$ and Map2PointPreImages$_\mathcal{G}$ for $\mathcal{G}_{p \cdot c}$ can be defined such that it returns all of the preimages of the c ambiguous representations of an input, which are bounded by $n_{\max} \cdot c$.

As ScalarMultVfy and ScalarMult use exponents that are a multiples of c they are guaranteed to produce a unique result on $\mathcal{G}_p$ for all of the c ambiguous representations of a point.

This change is compensated by the simulation by calling an experiment library using the ccExp class from Fig. 6. (Note that this class also accepts points on the quadratic twist, a feature that will become relevant only in the upcoming sections.) The ccExp object forwards queries to a $\mathcal{D}_{\mathcal{G}}$_sSDH object such that all inputs to the DDH oracle will be in $\mathcal{G}_p$.

Note that without exponents being a multiple of c, we would have had game $\mathbf{G}_3$ and $\mathbf{G}_4$ distinguishable because without the factor c in the secret scalars of honest parties, Y_a and Y_b could have nontrivial low-order components, which is noticeable and does occur in game $\mathbf{G}_4$. □

6.3 CPace using single-coordinate Diffie-Hellman

Some Diffie-Hellman protocols, including CPace, can be implemented also on a group modulo negation, i.e. a group where a group element Y and its inverse Y^{-1} (i.e. the point with $I = Y \cdot Y^{-1}$) are not distinguished and share the same binary representation. An elliptic curve in Weierstrass representation becomes a group modulo negation when only using x-coordinates as representation. We use the notation $\hat{Y}$ for such ambiguous encodings and use $\hat{Y} \leftarrow \mathsf{SC}(Y)$ for a function returning the x-coordinate for a point Y and $(Y^{-1}, Y) \leftarrow \mathsf{RC}(\hat{Y})$ for the inverse operation reconstructing Y and Y^{-1} in an undefined order. The major advantage of using this type of ambiguous encoding is that it can be helpful in practice for all of the following: reducing code size, reducing key sizes and network bandwidth, avoiding implementation pitfalls [Ber06] and restricting invalid curve attacks to the curve's quadratic twist. Consequently, many real-world protocols such as TLS only use this single coordinate for deriving their session key, as to give implementers the flexibility to take benefit of the above advantages.

We formalize CPace with single-coordinate DH, $\text{CPace}_{\mathsf{x-only}}$, by letting $\mathsf{ScalarMult}(\hat{Y}, y)$ and $\mathsf{ScalarMultVfy}(\hat{Y}, y)$ use the ambiguous encoding $\hat{Y}$.

Theorem 6.3 (Security of $\text{CPace}_{\mathsf{x-only}}$). *Assume CPace on a group $\mathcal{G}$ can distinguished from an ideal-world run of $\mathcal{F}_{\mathsf{lePAKE}}$ and $\mathcal{S}$ with at most negligible advantage, where $\mathcal{S}$ embeds an experiment object* exp *for the sSDH problem. Then $\text{CPace}_{\mathsf{x-only}}$ on the corresponding group modulo negation $\hat{\mathcal{G}}$ can be simulated by an ideal-world simulator $\hat{\mathcal{S}}$ that is obtained by chaining* exp *with* moduloNegationAdapter*, the adapter class from Fig. 7. The difference in the distinguishing advantage is bounded by a factor of 2.*

```
# using python-style notation with self pointer $s$
def class moduloNegationAdapter:
   "uses the strip- and reconstruct functions SC and RC."
   def _init_($s$,, baseExperiment):
      $s$.exp ←baseExperiment;$s$.records← [];
   def sampleY($s$):
      $Y \leftarrow ((s.\mathsf{exp}).\mathsf{sampleY}())^{s.c}$;
      $s$.records.append($Y$); return SC($Y$);
   def isValid($\hat{X}$):
      $(X_0, X_1)$←RC($\hat{X}$); return $(s.\mathsf{exp}).\mathsf{isValid}(X_0)$;
   def sampleH1($s$): return ($s$.exp).sampleH1();
   def corrupt($s, h, \hat{Y}$):
      $(Y, Y^*) \leftarrow \mathsf{RC}(\hat{Y})$; if $Y^*$ in $s$.records: $Y \leftarrow Y^*$;
      return ($s$.exp).corrupt($h, Y$);
   def DDH($s$,$G, \hat{Y}, \hat{X}, \hat{K}$):
      $(Y, Y^*) \leftarrow \mathsf{RC}(\hat{Y})$; if $Y^*$ in $s$.records: $Y \leftarrow Y^*$;
      $(X, X^*) \leftarrow \mathsf{RC}(\hat{X})$; $(K, K^*) \leftarrow \mathsf{RC}(\hat{K})$;
      return $(s.\mathsf{exp.DDH}(G, Y, X, K))$or$(s.\mathsf{exp.DDH}(G, Y, X, K^*))$

# Chaining the experiments for CPace on prime order curve,
# single coordinate, single map execution
sSdhExp = sSDH(sCDH);
distExp = $\mathcal{D}_{\mathcal{G}}$_sSDH(Map2PointPreImages, $n_{\mathsf{max}}$,sSdhExp);
singleCoorExp = moduloNegationAdapter(distExp)
```

Figure 7: Single-coordinate experiment class definition for CPace instantiations on groups modulo negation.

Proof Sketch. First note that the functions SC and RC that implement conversion between group and group modulo negations are both efficient, as in practice the most complex substep is a square root in $\mathbb{F}_q$. Secondly, CPace (except for instantiations using a map-twice-and-add construction) does not need a full group structure at any point. Instead, only chained exponentiations are used ($K = Y_a^{y_b} = Y_b^{y_a} = G^{y_a \cdot y_b}$), which can likewise be implemented on a group modulo negation. The protocol's correctness is not affected.

When starting with single-coordinate CPace in the real world, the same proof strategy applies, however the collision probability in game $\mathbf{G}_3$ is increased by a factor of 2. In game $\mathbf{G}_4$ no change is required. Also in games $\mathbf{G}_5$ and $\mathbf{G}_6$ the only difficulty shows up when splitting off the code for the full-group versions of the sSDH and sCDH experiments from the simulator code. We cannot embed the challenges as-is but need to reconstruct the sign information in order to serve the API of the full-group experiments. The corresponding strategy is implemented in the moduloNegationAdapter adapter class from Fig. 7. Note that this strategy never aborts itself but its incorporated sSDH or sCDH experiments abort upon DDH queries. As the moduloNegationAdapter adapter queries the DDH at most two times, the loss of $\text{CPace}_{\mathsf{x-only}}$ in comparison to CPace is at most a factor of two. □

6.4 CPace using twist secure curves

For a curve in Weierstrass form constructed over a field $\mathbb{F}_q$, a coordinate x represents either the x coordinate of a point on the curve itself or its so-called quadratic twist. For Diffie-Hellmann protocols where active adversaries are relevant, using single-coordinate Diffie-Hellman provides the advantage that the adversary may only insert inputs taken from one of these two curves, thus limiting the impact of invalid curve attacks if both, the curve and the twist, are appropriately designed. In this case, the computationally costly point verification algorithms can be avoided if the co-factor of $\mathcal{G}_{p \cdot c}$ is an integer multiple of the co-factor of the twist $\mathcal{G}'$ and all exponentiations clear the cofactor by choosing their secret exponents by a multiple of the cofactor c. This was first observed by Bernstein [Ber06] for Curve25519. This property can be employed also in the context of CPace for making implementations faster and more resilient against implementation errors regarding the point verification, specifically if CPace is implemented using scalar multiplication based on single-coordinate Montgomery ladders.

```
# using python-style notation with self pointer $s$
def class sCDH_sTCDH ():
   "Accepts $X, K$ inputs from $\mathcal{G}$ and the twist $\mathcal{G}$' "
   def _init_($s, G$):
      $s.B \leftarrow G; s.i \leftarrow 0; s.s_1, s.s_2 \leftarrow$ fresh;
   def sampleY(s):
      if $s.i \leq 2$:
         $s.i+ = 1$;sample $s.y_i \leftarrow_{\mathrm{R}} \mathbb{F}_p \setminus 0$;
         $s.Y_i \leftarrow (s.B)^{y_i}$; return $s.Y_i$;
   def corrupt(s,$Y$):
      for $1 \leq m \leq s.i$):
         if $(Y = (s.B)^{s.y_m})$:
            $x \leftarrow s.y_m; s.s_i \leftarrow$ corrupt; return x;
    def DDH($s, B, Y, X, K$):
      if ($\{Y, X\}$=$\{s.Y_1, s.Y_2\}$)and($s.s_1 =$ fresh)and($s.s_2 =$ fresh) ...
            ... and $(K = (s.B)^{s.y_1 \cdot s.y_2})$:
         abort("$K$ solves sCDH($B, Y_1, Y_2$)");
      if $(Y = s.Y_1)$:
         if($s.s_1 =$ fresh) and ($X \in \mathcal{G}' \setminus I_{\mathcal{G}'}$) and ($X^{s.y_1} = K$):
            abort("$X, K$ solve sTCDH($B, Y_1$)");
         return ($X^{s.y_1} = K$);
      if $(Y = s.Y_2)$:
         if($s.s_2 =$ fresh) and ($X \in \mathcal{G}' \setminus I_{\mathcal{G}'}$) and ($X^{s.y_2} = K$):
            abort("$X, K$ solve sTCDH($B, Y_2$)");
         return ($X^{s.y_2} = K$);
    def isValid($X$): return ($X \in (\mathcal{G} \setminus I_{\mathcal{G}})$) or ($X \in (\mathcal{G}' \setminus I_{\mathcal{G}'}$);

# Chaining the experiment objects for case of X25519 and X448
sSdhExp = sSDH(sCDH_sTCDH);
ccExp = cofactorClearer(sSdhExp,$c, p, p_{twist}$);
distExp = $\mathcal{D}_{\mathcal{G}}$_sSDH(Map2PointPreImages, $n_{\max}$,ccExp);
twistSecExp =moduloNegationAdapter(ccExp);
```

Figure 8: Class combining sCDH and sTCDH experiments

Theorem 6.4 (Security of CPace on twist-secure curves.). *Let $\mathcal{G}_{p \cdot c}$ be an elliptic curve of order $c \cdot p$ where c, p coprime with a subgroup $\mathcal{G}$ of prime order p over base field F_q. Let $\mathcal{G}_{p' \cdot c'}$ be the quadratic twist of $\mathcal{G}_{p \cdot c}$ of order $c' \cdot p_{twist}$ with a subgroup $\mathcal{G}'$ of order p_{twist}. Let c be equal to c' or an integer multiple of c'.*

If there is a single-coordinate implementation of CPace on $\mathcal{G}$ that could be distinguished from an ideal-world simulator $\mathcal{S}$ from Fig. 4 with negligible advantage, where $\mathcal{S}$ embeds an experiment object exp *for the sCDH problem, then a modified CPace implementation that does not discard received points from $\mathcal{G}' \setminus I_{\mathcal{G}'}$ can be simulated by an ideal-world simulator $\mathcal{S}'$ that is obtained by replacing the sCDH experiment by the experiment object twistSecExp from figure 8 and the difference in the distinguisher advantages is bounded by* $2 \cdot \mathbf{Adv}^{sTCDH}$.

Proof. The only difference to the proof strategy for Theorem 5.1 consists in the fact that we now need to handle events where the adversarial strategy is based on injecting points from the twist.

First note that the twist has cofactor c' and thus c' low-order points. As c is an integer multiple of c' the exponentiation by multiples of c maps all low-order points on the twist to the twist's identity element and these will be discarded also by the modified CPace version. In the simulation we additionally need to handle the case that the adversary provides an input from the twist. We add an additional game after $\mathbf{G}_0$, where we abort if the adversary queries H_2 for Y, X, K for a honest $Y = G^y$ calculated from a generator G, an adversarial input X from the twist $\mathcal{G}'$ such that $K = X^y \in \mathcal{G}'$, where y is the private scalar of one of the honest parties.

This change is distinguishable only if $\mathcal{A}$ provided a solution to the sTCDH problem from Definition 3.7 and the advantage of a distinguisher of in this additional game is bounded by $\mathbf{Adv}^{\mathrm{sTCDH}}$. In any other case the honest party will return ISK values indistinguishable from random values in both, the real world and the ideal world. As the adversary may choose to attack both honest parties, the advantage has to be multiplied by two. We incorporate this change in the simulator by replacing the sCDH object instance that is embedded by the sSDH experiment class. □

6.5 Chaining the experiment classes

In Appendix F, we describe 3 CPace implementations which combine the single aspects discussed in the previous sections in various combinations. The experiment that encodes the assumptions that apply for the construction that we recommend for use on twist secure Montgomery curves is specified by the "twistSecExp" object from Fig. 8. Correspondingly the assumption set that is necessary for proving the security of CPace on the group abstraction construction from Appendix F for ristretto25519 and decaf448 [Ham15a, dVGT+20] is defined by the "coffeeExp" object from Fig. 11. The assumption set needed for proofs for our recommended construction with short-Weierstrass curves is specified by the "singleCoorExp" object in figure Fig. 7. A concrete example how this process is carried out is given in Appendix D.

7 CONCLUSION

We demonstrated that the symmetric PAKE protocol CPace enjoys strong composable security guarantees under CDH-type assumptions, and in the random oracle model. We identify a requirement we coin *probabilistic invertibility* on the Map2Point primitive being employed by CPace for deriving secret generators, such that the random-oracle idealization can be restricted to modeling conventional hash functions only. All Map2Pointfunctions currently considered by the CFRG working group fulfill this new property. We believe that this is the first work that goes beyond a modeling of Map2Point as a random oracle for a real-world cryptographic protocol.

In the process of analyzing CPace we noticed a significant shortcoming of all previously published ideal UC PAKE functionalities. We presented and used a strengthened functionality that will provide major advantages when analyzing protocols that use CPace, and PAKE in general, as a building block.

We presented a flexible and modular approach for carrying out security proofs in simulation-based models when reducing to assumptions for which experiment algorithms can be efficiently implemented. The approach consists of making the challenges an integral part of the simulation algorithm. This avoids the unnatural separation between simulator construction and reduction arguments. We have employed this approach for deriving the exact changes in the assumption set when fine-tailoring CPace for different elliptic-curve ecosystems such as short Weierstrass curves, Montgomery curves and idealized group abstractions. We were able to prove security of many such CPace variants even in conjunction with adaptive corruption and believe that CPace is the first PAKE protocol for which security guarantees are proven for such a strong adversary model.

To conclude, we believe that our techniques can also be employed for more closely understanding the security properties of other protocols operating on elliptic curves in conjunction with Map2Point primitives such as, e.g., oblivious pseudo-random functions (OPRF) that form building blocks of OPAQUE [JKX18] and strong AuCPace [HL18]. Current analyses model the mapping primitive as a random oracle. With our work as motivating and technical example, a security proof of more efficient and less complex OPRF constructions based on single-coordinate Montgomery ladders seems within reach. Secondly, we believe that our proof strategy for assumptions with efficient experiments can provide advantages for more flexibly analyzing different variants of a protocol, and might even open easier paths for employing machine-based proof strategies in UC.

ACKNOWLEDGMENTS

This work was partly supported by the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No. 786725 – OLYMPUS.

REFERENCES

[ABB+20a] Michel Abdalla, Manuel Barbosa, Tatiana Bradley, Stanislaw Jarecki, Jonathan Katz, and Jiayu Xu. Universally composable relaxed password authenticated key exchange. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 278–307. Springer, Heidelberg, August 2020.

[ABB+20b] Michel Abdalla, Manuel Barbosa, Tatiana Bradley, Stanislaw Jarecki, Jonathan Katz, and Jiayu Xu. Universally composable relaxed password authenticated key exchange. Cryptology ePrint Archive, Report 2020/320, 2020. https://eprint.iacr.org/2020/320.

[ABK+20] Michel Abdalla, Manuel Barbosa, Jonathan Katz, Julian Loss, and Jiayu Xu. Algebraic adversaries in the universal composability framework. Pre-Print, October 2020.

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle diffie-hellman assumptions and an analysis of DHIES. In David Naccache, editor, *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2001.

[AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 191–208. Springer, Heidelberg, February 2005.

[BCI+10a] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 237–254. Springer, 2010.

[BCI+10b] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 237–254. Springer, Heidelberg, August 2010.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 207–228. Springer, Heidelberg, April 2006.

[BFK09] Jens Bender, Marc Fischlin, and Dennis Kügler. Security analysis of the PACE key-agreement protocol. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, editors, *ISC 2009*, volume 5735 of *LNCS*, pages 33–48. Springer, Heidelberg, September 2009.

[BHKL13] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 967–980. ACM, 2013.

[BL19] Daniel J. Bernstein and Tanja Lange. SafeCurves: Choosing safe curves for elliptic-curve cryptography. Definition of Twist security. (accessed on 15 January 2019), 2019. https://safecurves.cr.yp.to/twist.html.

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, Heidelberg, May 2005.

[DSS13] Digital Signature Standard (DSS). National Institute of Standards and Technology (NIST), FIPS PUB 186-4, U.S. Department of Commerce, July 2013. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

[dVGT+20] H. de Valence, J. Grigg, G. Tankersley, F. Valsorda, I. Lovecruft, and M. Hamburg. The ristretto255 and decaf448 groups. Rfc, IRTF, 10 2020.

[ECC18] Elliptic Curve Cryptography. Federal Office for Information Security (BSI), Technical Guideline BSI TR-03111, Version 2.10, June 2018.

[FHSS+19] A. Faz-Hernandez, S. Scott, N. Sullivan, R. Wahby, and C. Wood. Hashing to elliptic curves, 2019. https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve/.

[FT12] Pierre-Alain Fouque and Mehdi Tibouchi. Indifferentiable hashing to Barreto-Naehrig curves. In Alejandro Hevia and Gregory Neven, editors, *LATINCRYPT 2012*, volume 7533 of *LNCS*, pages 1–17. Springer, Heidelberg, October 2012.

[Haa20] Björn Haase. CPace, a balanced composable PAKE, 2020. https://datatracker.ietf.org/doc/draft-haase-cpace/.

[Ham15a] Mike Hamburg. Decaf: Eliminating cofactors through point compression. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 705–723. Springer, 2015.

[Ham15b] Mike Hamburg. Ed448-goldilocks, a new elliptic curve. Cryptology ePrint Archive, Report 2015/625, 2015. http://eprint.iacr.org/2015/625.

[Ham20] Mike Hamburg. Indifferentiable hashing from elligator 2. Cryptology ePrint Archive, Report 2020/1513, 2020. https://eprint.iacr.org/2020/1513.

[Hes20] Julia Hesse. Separating symmetric and asymmetric password-authenticated key exchange. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 579–599. Springer, Heidelberg, September 2020.

[HL18] Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. Cryptology ePrint Archive, Report 2018/286, 2018. https://eprint.iacr.org/2018/286.

[HL19] Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR TCHES*, 2019(2):1–48, 2019. https://tches.iacr.org/index.php/TCHES/article/view/7384.

[HWCD08] Hüseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Edward Dawson. Twisted edwards curves revisited. In *Asiacrypt*, volume 5350, pages 326–343. Springer, 2008.

[Jab96] David P. Jablon. Strong password-only authenticated key exchange. *Computer Communication Review*, 26(5):5–26, 1996.

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 456–486. Springer, Heidelberg, April / May 2018.

[LHT16] A. Langley, M. Hamburg, and S. Turner. Elliptic Curves for Security. RFC 7748, IETF, January 2016.

[LM10] Manfred Lochter and Johannes Merkle. Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation. RFC 5639, IETF, March 2010.

[PAC08] Advanced security mechanism for machine readable travel documents (extended access control (EAC), password authenticated connection establishment (PACE), and restricted identification (RI)). Federal Office for Information Security (BSI), BSI-TR-03110, Version 2.0, 2008.

[PW17] David Pointcheval and Guilin Wang. VTBPEKE: Verifier-based two-basis password exponential key exchange. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *ASIACCS 17*, pages 301–312.

ACM Press, April 2017.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. *IACR Cryptol. ePrint Arch.*, 2004:332, 2004.

[SvdW06] Andrew Shallue and Christiaan E. van de Woestijne. Construction of rational points on elliptic curves over finite fields. In *ANTS*, volume 4076 of *Lecture Notes in Computer Science*, pages 510–524. Springer, 2006.

A PROOF OF THEOREM 5.1

We start with the real execution of the CPace protocol with an adversary $\mathcal{A}$, and gradually modify it, ending up with the ideal execution $\mathcal{F}_{\mathsf{lePAKE}}$ with a simulator $\mathcal{S}$. The changes will go unnoticed by an (adaptively corrupting) environment $\mathcal{Z}$ interacting with parties and the adversary. Let $\mathrm{Real}_{\mathcal{Z}}(CPace, \mathcal{A})$ be the event that environment $\mathcal{Z}$ with adversary $\mathcal{A}$ and an execution of CPace outputs 1 and $\mathrm{Ideal}_{\mathcal{Z}}(\mathcal{F}_{\mathsf{lePAKE}}, \mathcal{S})$ be the corresponding event in the ideal execution with functionality $\mathcal{F}_{\mathsf{lePAKE}}$ depicted in Fig. 1. We assume $\mathcal{S}$ to always include the session identifier *sid* in each stored record, and omit them in this proof for brevity.

Game $\mathbf{G}_0$: The real protocol execution. This is the real world in which the adversary interacts with real players and may view, modify and/or drop network messages and adaptively corrupt parties.

$$Pr[\mathrm{Real}_{\mathcal{Z}}(\mathrm{CPace}, \mathcal{A})] = Pr[\mathbf{G}_0]$$

Game $\mathbf{G}_1$: Introducing the simulator. In this game we move the whole execution into one machine and call in the simulator $\mathcal{S}$. Note that this implies that $\mathcal{S}$ implements the random oracles $\mathsf{H}_1, \mathsf{H}_2$ and runs the execution with actual passwords as input. We will change the simulation to work without passwords in the upcoming games. In this game for any new input query s for H_1 (H_2) samples a point on the curve (a random string) respectively. No re-programming operation is yet needed. The changes are only syntactical and thus

$$\Pr\,[\mathbf{G}_0\,] = \Pr\,[\mathbf{G}_1\,]$$

Game $\mathbf{G}_2$: Embedding trapdoors. In this game we start keeping track of secret exponents for generators created from passwords. $\mathcal{S}$ samples a fixed generator $B \leftarrow \mathcal{G}$. For every hash query $\mathsf{H}_1(sid, \mathcal{P}, \mathcal{P}', \mathrm{pw})$, $\mathcal{S}$ samples $r \leftarrow \mathbb{F}_p$, stores $(\mathsf{H}_1, \mathcal{P}||\mathcal{P}', \mathrm{pw}, r, r^{-1}, B^r)$ and replies to the query with B^r. The only difference is that the simulator now keeps track of the secret exponents for H_1 queries and the distributions of this and the previous game are perfectly indistinguishable

$$\Pr\,[\mathbf{G}_1\,] = \Pr\,[\mathbf{G}_2\,]$$

Game $\mathbf{G}_3$: Abort on collisions of the random oracle H_1. The simulator aborts if a collision occurs in H_1, i.e., if $\mathcal{S}$ samples an answer for a fresh H_1 query that he already gave before. Note that without collisions two honest parties will always output matching (respectively differing) session keys if both, passwords and party identifiers, match (respectively differ). Note that authenticating party identifiers in addition to the password is important for fending off relay attacks. As the order of the group is $p \approx 2^k$ and $\mathcal{Z}$ can only make a polynomially bounded number l_{H_1}) of H_1 queries, the probability of aborts is negligible in k by the birthday bound. Thus this and the previous game are indistinguishable.

$$|\Pr\,[\mathbf{G}_2\,] \text{ - } \Pr\,[\mathbf{G}_3\,]\,| \le l_{\mathsf{H}_1}^2/p$$

Game $\mathbf{G}_4$: Introduce $\mathcal{F}$ and simulate the protocol messages. In the real world all session keys will be calculated by using H_2 queries. In the ideal world session keys of honest parties will not be generated by H_2 queries but will be provided separately by the ideal functionality (with mechanisms in $\mathcal{F}$ for synchronizing H_2 outputs and session keys in case of successful password guesses). Here we prepare this separation and use programming operations for the H_2 RO for keeping session key outputs consistent with H_2.

Changes to the functionality. In this game we add an ITI $\mathcal{F}$ external to the simulator. $\mathcal{F}$ has all interfaces of $\mathcal{F}_{\mathsf{lePAKE}}$ except that we don't yet limit the number of calls to the LateTestPwd and TestPwd queries and make the NewKey interface always relay keys coming from the simulator via NewKey queries. Also in this game we let $\mathcal{F}$ inform the simulator about the clear-text passwords upon NewSession events.

Changes to the simulation. Upon receiving an adversarially generated $Y_a \in \mathcal{G} \setminus I_{\mathcal{G}}$ or $Y_b \in \mathcal{G} \setminus I_{\mathcal{G}}$ aimed at a party $\mathcal{P}$, $\mathcal{S}$ sends $(\mathsf{RegisterTest}, \mathcal{P})$ to $\mathcal{F}$. $\mathcal{S}$ simulates protocol messages $Y_a = B^{z_a}$ and $Y_b = B^{z_b}$ on behalf of honest parties by sampling exponents z_a, z_b uniformly from $(1 \cdots (p-1))$. Upon an adaptive corruption query of $\mathcal{Z}$ for a party outputing Y_a (respectively Y_b), $\mathcal{S}$ converts these to the secret scalars y_a (respectively y_b) used by the real-world protocol as $y_a = z_a \cdot r^{-1}$ (respectively $y_b = z_b \cdot r^{-1}$), where r^{-1} has been looked up in the record $(\mathsf{H}_1, \mathrm{oc}(\mathcal{P}, \mathcal{P}'), \mathrm{pw}, *, r, r^{-1}, G)$ for the party's password. ($\mathcal{S}$ creates a new such record for pw if none yet exists.) As $r \neq 0$ uniformly sampling y_a (y_b) or z_a ($z_b y$) is equivalent. In the following we detail how the simulator produces consistent outputs and answers to H_2 queries. While being straightforward, the presentation is slightly involved since the order of queries and messages impacts the simulation.

When $\mathcal{S}$ needs to issue a key to a party $\mathcal{P}$ using password pw for points Y_a and Y_b, then either Y_a or Y_b will have been generated on behalf of $\mathcal{P}$. $\mathcal{S}$ looks for a record $(\mathsf{H}_1, \mathrm{oc}(\mathcal{P}, \mathcal{P}'), \mathrm{pw}, *, r, r^{-1}, G)$ (and creates a H_1 entry for pw if no such record exists). If Y_a was generated for $\mathcal{P}$, then set $K' = Y_b^{z_a r^{-1}}$, otherwise set $K' = Y_a^{z_b r^{-1}}$.

- *Adjust output of $\mathcal{P}$ to earlier H_2 query:* $\mathcal{S}$ then checks whether there is any record $(\mathsf{H}_2, K||\mathrm{oc}(Y_a, Y_b), ISK)$ such that $K = K'$. Note that in this case it holds that both, $DDH(B^r, Y_a, Y_b, K) = DDH(B, Y_a, Y_b, K^{r^{-1}}) = 1$. In this case $\mathcal{S}$ executes the TestPwd query of $\mathcal{F}$ for (pw) and subsequently passes ISK to the NewKey query of $\mathcal{F}$. We note that our simulator does not make use of the reply "correct/wrong guess" (since there are no more values of the honest party to simulate after the guess happens), but still needs to issue TestPwd in order to be able to determine the attacked party's output via NewKey in case of a successful guess.
- *Align keys in case of matching passwords:* If no corresponding H_2 record is found and both points Y_a, Y_b were generated by honest parties and the passwords of both parties match and a NewKey query has already been issued to

the other party, then $\mathcal{S}$ calls NewKey for $\mathcal{P}$ using the key already passed to $\mathcal{P}'$.

- In any other case there is no output yet to keep consistent with. $\mathcal{S}$ samples a new random key just as for new H_2 queries and passes this key to the NewKey query of $\mathcal{F}$.

Upon $\mathcal{Z}$ querying $\mathsf{H}_2(K||Y_a||Y_b)$ for a yet unqueried input,

- *Query not related to any honest output:* If neither Y_a nor Y_b were generated by an honest party or $Y_a||Y_b \neq (Y_a|| < ||Y_b)$ then sample a new random value and output it as result for H_2.
- *Adjust* H_2 *query to key of* $\mathcal{P}$*:* Else if all of (1) $Y_a = B^{z_a}$ was simulated for honest party $\mathcal{P}$ and (2) NewKey was already delivered to $\mathcal{F}$ for $\mathcal{P}$ and (3) there is a record $(\mathsf{H}_1, \mathrm{oc}(\mathcal{P}, \mathcal{P}'), \mathrm{pw}, r, r^{-1}, G)$ such that $K = Y_b^{z_a r^{-1}}$ (which occurs if $DDH(B^r, Y_a, Y_b, K) = DDH(B, Y_a, Y_b, K^{r^{-1}}) = 1$) then: $\mathcal{S}$ sends $(\mathsf{LateTestPwd}, \mathcal{P}, \mathrm{pw})$ to $\mathcal{F}$. $\mathcal{S}$ programs $\mathcal{F}$'s answer to this query as reply to the H_2 query.
- *Adjust* H_2 *query to key of* $\mathcal{P}'$*:* Else if $Y_b = B^{z_b}$ was simulated for honest party $\mathcal{P}'$. $\mathcal{S}$ proceeds as for honest $\mathcal{P}$ but checks for $K = Y_a^{z_b r^{-1}}$.
- Else we conclude that there is no need for adjustment and the H_2 query is answered as in the previous game.

Note that with these changes, $\mathcal{S}$ never needs to actually calculate itself a Diffie-Hellman result point K, instead it only makes sure that H_2 queries for parties that adhere to the protocol and keys output to honest parties match. Instead from this point on, the simulators will only need access to $DDH(B, Y, \cdot, \cdot)$ or $DDH(B^r, Y, \cdot, \cdot)$ oracles for a fixed generator B and a honestly generated point Y as second parameter which we could implement easily in this game as we have access to the secret exponents.

Indistinguishability argument. There is no change from the viewpoint of $\mathcal{Z}$ between this and the previous game. The game only differs in the way how session keys and H_2 queries are output. In both games, both, the output of H_2 queries and session keys, are uniformly sampled. Just as in the previous game, session keys output to two honest parties match if $\mathcal{P}$ and $\mathcal{P}'$ have used same password and differ otherwise.

Finally the session keys output to honest parties with password pw match H_2 queries taken for queries using a Diffie-Hellman point K that would be calculated by parties that follow the protocol. It follows that

$$\Pr\left[\mathbf{G}_3\ \right] = \Pr\left[\mathbf{G}_4\ \right]$$

Game $\mathbf{G}_5$: Limit number of password guesses

Changes to the functionality. The number of password guesses is now limited to one guess per party. I.e. there is one guess per record $\langle sid, \mathcal{P}, \mathcal{P}', \mathrm{pw}, \cdot \rangle$.

Changes to the simulation. $\mathcal{S}$ looks for CDH tuples in H_2 queries w.r.t all recorded exponents r in its H_1 list. If $\mathcal{S}$ finds a CDH tuple (G, X, Y, K) that meets the conditions of $\mathbf{G}_4$ (where Y denotes the adversarially-generated message), $\mathcal{S}$ creates a record $(\mathtt{guess}, G, Y)$. If at this point there is already a record $(\mathtt{guess}, G', Y)$ with $G \neq G'$, we say that event `multguess` happens and let $\mathcal{S}$ abort.

Indistinguishability argument. The change is only recognizable if $\mathcal{S}$ has to abort. We show that this happens only with negligible probability if the sSDH assumption holds in $\mathcal{G}$. We construct an efficient sSDH adversary $\mathcal{B}_{\mathsf{sSDH}}$ interacting with $\mathcal{Z}$. Let (Y, G_1, G_2) denote a sSDH challenge. $\mathcal{B}_{\mathsf{sSDH}}$ embeds the challenge in this game as follows: first, $\mathcal{B}_{\mathsf{sSDH}}$ flips a coin and sets Y to be either the message of $\mathcal{P}$ or $\mathcal{P}'$. $\mathcal{B}_{\mathsf{sSDH}}$ aborts if the chosen party is corrupted or gets corrupted at a later stage. Then, $\mathcal{B}_{\mathsf{sSDH}}$ randomly chooses two out of all H_1 queries made by $\mathcal{Z}$ and answer them with $G_1 \leftarrow \mathsf{H}_1(sid||\mathrm{oc}(\mathcal{P}, \mathcal{P}')||\mathrm{pw})$ and $G_2 \leftarrow \mathsf{H}_1(sid||\mathrm{oc}(\mathcal{P}, \mathcal{P}')||\mathrm{pw}')$ (pw and pw′ are going to be $\mathcal{Z}$'s two password guesses). $\mathcal{B}_{\mathsf{sSDH}}$ replaces the check performed by the simulator with oracle queries $DDH(G_i, Y, Y', K)$, $i = 1, 2$, where Y' denotes the (simulated or adversarial) other message. If `multguess` occurs, then two CDH solutions were found, namely one for each G_i. $\mathcal{B}_{\mathsf{sSDH}}$ outputs these two solutions. Since w.l.o.g $\mathcal{Z}$ corrupts at most one party and has a view independent of the coin flipped by $\mathcal{B}_{\mathsf{sSDH}}$, $\mathcal{B}_{\mathsf{sSDH}}$ has to abort only with probability 1/2. Overall, it follows that $Pr[\mathtt{multguess}] \leq 2l_{\mathsf{H}_1}^2 \mathbf{Adv}^{\mathsf{sSDH}}$, where l_{H_1} denotes the number of H_1 queries made by $\mathcal{Z}$. We thus have

$$\mid \Pr\left[\mathbf{G}_4\ \right] - \Pr\left[\mathbf{G}_5\ \right] \mid \leq 2l_{\mathsf{H}_1}^2 \mathbf{Adv}^{\mathsf{sSDH}}\ .$$

Remark A.1. *We note that any reduction could make use of the true password of* $\mathcal{P}'$*, since a reduction interfaces with* $\mathcal{Z}$ *and thus also receives protocol inputs. However, it is unclear how to leverage this, since* `multguess` *implies that* $\mathcal{Z}$ *makes at least one incorrect guess. Thus,* $\mathcal{S}$ *needs to turn password guesses into CDH solutions* regardless *of whether a guess is correct or not. Further, we note that a reduction to the strong CDH assumption seems infeasible here. The reason is that* $\mathcal{S}$ *needs to detect password guesses in* H_2 *queries, which requires knowledge of the exponent of the simulated message* Y_b*. The reduction however does not have this knowledge due to* Y_b *being set to the CDH challenge, and thus needs to leverage DDH oracles w.r.t different generators for detecting both guesses.*

Game $\mathbf{G}_6$: Random key if passwords mismatch

Changes to the functionality. In case a record is interrupted, $\mathcal{F}$ now outputs a random key (instead of the one given by the simulator via NewKey).

Changes to the simulation. -

Indistinguishability argument. Assume the output was generated for an honest $\mathcal{P}$ (the other case works analogously). The output towards $\mathcal{Z}$ differs only in case TestPwd returns "wrong guess" or LateTestPwd returns a random key. $\mathcal{S}$ only issues these queries if he finds a CDH tuple in H_2. In case of TestPwd (Y_b was sent after H_2 query), let Y_a, Y_b denote the transcript and K the Diffie-Hellman value computed by $\mathcal{P}$. The environment never submits $(K||\mathrm{oc}(Y_a, Y_b))$ to H_2, as otherwise the previous game would abort due to `multguess` happening. Thus, the output $\mathsf{H}_2(K||\mathrm{oc}(Y_a, Y_b))$ of $\mathcal{P}$ in $\mathbf{G}_5$ is uniformly random from the viewpoint of $\mathcal{Z}$, and replacing it with a fresh random output chosen by $\mathcal{F}$ due to the interrupted record is perfectly indistinguishable.

In case of LateTestPwd (H_2 was queried after $\mathcal{P}$ generated output), the key output to $\mathcal{P}$ was the randomly chosen K generated by the LateTestPwd interface of $\mathcal{F}_{\mathsf{lePAKE}}$ in the previous game, which is perfectly indistinguishable from the randomly chosen one that the NewKey interface of $\mathcal{F}_{\mathsf{lePAKE}}$ outputs directly to $\mathcal{P}$ in this game due to the record being interrupted.

Game $\mathbf{G}_7$: Output random keys for honest sessions

In this game, we let the functionality generate parties' outputs in honest sessions. We change the simulation to work without passwords and without knowledge of the honest parties outputs.

Changes to the ideal functionality. We now add the full NewKey interface to $\mathcal{F}$.

Changes to the simulation. Let m denote the number of H_1 queries issued by $\mathcal{Z}$ and $r_1, \ldots r_n \leftarrow \mathbb{F}_q$ the trapdoors embedded in these queries (see $\mathbf{G}_2$). Let z_a, z_b denote the exponents of simulated messages as of $\mathbf{G}_4$. In case $\mathcal{Z}$ queries $\mathsf{H}_2(K||\mathsf{oc}(Y_a, Y_b))$, where for some $i \in [m]$ (B, Y_a, Y_b, K^{r_i}) is a CDH tuple (which can be checked by $\mathcal{S}$ via $B^{z_a z_b} = K^{r_i}$), $\mathcal{S}$ aborts.

Indistinguishability argument. First note that $\mathcal{Z}$ can only note a difference between this and the previous game if it reproduces any value K or K' computed by some (honest) party in a "fresh" and honest session. In particular, this means that $\mathcal{Z}$ cannot corrupt a party nor inject messages (as in this case $\mathcal{S}$ issues TestPwd or LateTestPwd and records get interrupted or compromised). Let us detail what party $\mathcal{P}$ computes (argument for $\mathcal{P}'$ is analogously). W.l.o.g we assume that $\mathcal{Z}$ queried $\mathsf{H}_1(sid||\mathsf{oc}(\mathcal{P}, \mathcal{P}')||\mathsf{pw})$, where pw is the password of $\mathcal{P}$, and obtained B^a as answer. $\mathcal{P}$ was simulated with values B, y_a, and we now implicitly adjust this to $B^a, y_a \cdot 1/a$. This lets $\mathcal{P}$ compute $K \leftarrow Y_b^{y_a 1/a} = B^{y_b y_a 1/a}$, where (B, Y_a, Y_b, K^a) is a CDH tuple. We stress that P computes the same value K regardless of whether both parties use matching or mismatching passwords, since $\mathcal{P}$'s output only depends on the simulated Y_b and is independent of the generator used by $\mathcal{P}'$.

We show that if sCDH holds in $\mathcal{G}$ then $\mathcal{S}$ never aborts. Consider the following efficient adversary $\mathcal{A}_{\mathsf{sCDH}}$. $\mathcal{A}_{\mathsf{sCDH}}$ obtains an sCDH challenge (B, Y_a, Y_b) and executes the simulation of game $\mathbf{G}_7$ with it. Upon Z querying $\mathsf{H}_2(K||\mathsf{oc}(Y_a, Y_b))$, $\mathcal{A}_{\mathsf{sCDH}}$ needs to detect CDH solutions using his own oracle instead of knowledge of exponents of Y_a, Y_b. For this, $\mathcal{A}_{\mathsf{sCDH}}$ calls $b \leftarrow \mathsf{DDH}(B, Y_a, Y_b, K^{a_i})$ and outputs K^{a_i} as sCDH solution if $b = 1$ happens. It follows that

$$| \Pr [\mathbf{G}_6] \text{ - } \Pr [\mathbf{G}_7] | \leq \mathbf{Adv}^{\mathsf{sCDH}} .$$

Game $\mathbf{G}_8$: Remove passwords from simulation.

Changes to the ideal functionality. We remove passwords from NewSession queries sent from $\mathcal{F}$ to $\mathcal{S}$.

Changes to the simulation. As the simulation already is independent of the password, there are no further changes required.

Since we are just removing unused values from the output of $\mathcal{F}$ towards $\mathcal{S}$, the output distributions towards $\mathcal{Z}$ of this and the previous game are indistinguishable. Hence,

$$Pr[\mathbf{G}_7] = Pr[\mathbf{G}_8] = Pr[\mathrm{Ideal}_{\mathcal{Z}}(\mathcal{F}_{\mathsf{lePAKE}}, \mathcal{S})]$$

This game is identical to the ideal execution since $\mathcal{F} = \mathcal{F}_{\mathsf{lePAKE}}$, which concludes the proof. The simulator of this final game is depicted in Fig. 3

B ON SHORTCOMINGS OF UC PAKE FUNCTIONALITIES

We provide an illustrating example on how a shortcoming in existing PAKE functionalities from the literature [CHK+05, JKX18, Hes20, ABB+20a] impacts their suitability for building higher-level applications from PAKE. Let us first detail what shortcoming we are talking about. In PAKE functionalities, the adversary $\mathcal{A}$ gets to determine the session key of honest users in some cases. This makes sense if $\mathcal{A}$ manages to guess an honest party's password during an interaction with said party, in which case $\mathcal{A}$ can compute the very same session key and thus the party's output is no longer uniformly random (from the viewpoint of the adversary). But strangely, all existing PAKE functionalities also allow the adversary to determine the honest party's key K^* via NewKey queries if (cf. Fig. 1)

either $\mathcal{P}$ or $\mathcal{P}'$ is corrupted.

This contradicts the principle of authenticated key exchange, where unauthenticated entities (i.e., honest or malicious parties not knowing the password) should not be allowed to learn the key computed by an honest party.

Does this mean all known PAKE protocols are insecure? Most importantly, all PAKE protocols proven w.r.t any of the existing PAKE functionalities *can still be considered secure.* On a technical level, the reason is that these functionalities still provide all guarantees one would expect from a PAKE against network attackers, as long as no corruptions occur. Besides that, we are not aware of any actual UC PAKE security analysis that exploits the above shortcoming in their simulation, and conjecture that they can be proven secure without this shortcoming.

Why bother then to fix this? The shortcoming's effect shows when a PAKE functionality is used to modularly build other protocols. Modular protocol analysis requires strong composability guarantees of security proofs and is one of the main features of the UC framework. As an example, assume we want to build password-authenticated secure channels from $\mathcal{F}_{\mathsf{pwKE}}$ depicted in Fig. 1. Intuitively, a password-authenticated secure channel allows two parties to securely communicate if *and only if* they hold the same password. Consider the following password-based channel toy protocol[1] Π_{sc} depicted in Fig. 9: users call $\mathcal{F}_{\mathsf{pwKE}}$ to turn their passwords into a cryptographic key and subsequently encrypt a message m under this key using a symmetric cipher.

Intuitively, we would expect protocol Π_{sc} to implement a secure password-authenticated channel. Unfortunately, with the shortcoming in existing PAKE functionalities, there is no way to prove this protocol secure: upon corrupting $\mathcal{P}'$, the adversary gets to determine the value K send to an honest $\mathcal{P}$. A simulator would now have to produce a ciphertext c that, for any K chosen by the

[1] This is just a demonstrating example and not a suggestion for a practical protocol, which would require authenticated encryption [CHK+05].

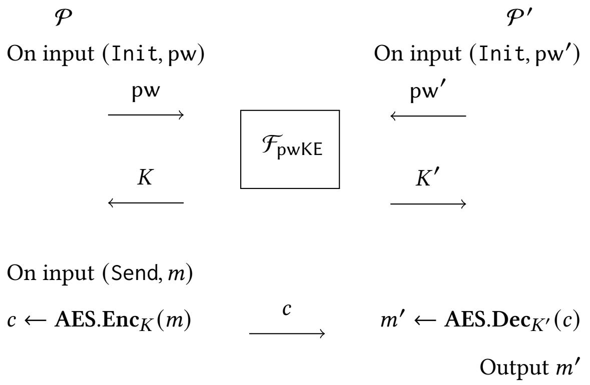

Figure 9: Toy example of a modular password-authenticated secure channel protocol with $\mathcal{F}_{\text{pwKE}}$ as building block, exposing the shortcoming of $\mathcal{F}_{\text{pwKE}}$.

CPace$_{\text{base}}$ is parametrized by a security parameter k and operates on a group $\mathcal{G}$ of order p that comes with (1) a scalar-multiply and (2) a scalar-multiply-and-verify function that responds with the neutral element $I_{\mathcal{G}}$ for invalid inputs.
Let $\mathsf{H}_1 : \{0,1\}^* \to \mathcal{G}$ and $\mathsf{H}_2 : \{0,1\}^* \to \{0,1\}^*$ be two hash functions. Let SampleScalar() denote an algorithm for random sampling from $\mathbb{F}_p \setminus \{0\}$.

Protocol:

(1) When $\mathcal{P}$ receives input (NewSession, $sid, \mathcal{P}, \mathcal{P}'$, pw), it calculates $G \leftarrow \mathsf{H}_1(sid||\text{oc}(\mathcal{P},\mathcal{P}')||\text{pw})$. It then samples a fresh nonzero scalar $y \leftarrow$ SampleScalar() calculates $Y \leftarrow \text{ScalarMult}(G, y)$. It sets up a session record $(sid, \mathcal{P}', y, Y, \text{fresh})$ and marks it as fresh and then it sends a (RemotePoint, $sid, \mathcal{P}, Y$) message to $\mathcal{P}'$.

(2) When $\mathcal{P}$ receives message (RemotePoint, $sid, \mathcal{P}', X$) and finds a fresh session record $(sid, \mathcal{P}', y, Y, \text{fresh})$ it then calculates $K \leftarrow \text{ScalarMultVfy}(X, y)$. If $K = I_{\mathcal{G}}$ it aborts. If $K \neq I_{\mathcal{G}}$, it calculates $ISK \leftarrow \mathsf{H}_2(sid||K||\text{oc}(X,Y))$, erases pw, G, y and K from its memory and outputs (sid, ISK). In either case mark the session record as completed.

Figure 10: UC execution of a simplified variant of CPace, CPace$_{\text{base}}$, proven secure in Theorem 5.1.

simulator, decrypts to m – but without knowing m. Clearly, this traps the simulator.

Lastly, we note that our argument above is backed up by Canetti et al. [CHK$^+$05], who could only circumvent the above simulation trap by integrating the shortcoming also into their password-based channel functionality.

To summarize, it seems necessary to strengthen UC PAKE functionalities in the way we propose in this paper, in order to make them useful as building blocks for higher level applications.

C INITIATOR-RESPONDER AND PARALLEL CPACE PROTOCOL VARIANTS

The current CPace specification in [Haa20] describes a protocol with clear initiator and responder roles, where the responder sends his reply only upon reception of the initiator message. Astonishingly, we observed that for an analysis of such a protocol in the UC framework, an ideal PAKE functionality technically needs a richer structure than the one for a corresponding *parallel* protocol that does not enforce ordering. The reason is that the responder party needs to be activated twice, once for sending the network message and once for issuing the session key.

```
# using python-style notation with self pointer $s$
def class mapTwice_sSDH:
  def _init_($s$, Map2PointPreImages, $n_{\max}$, sSDHExp):
    $s.sSDH$ = sSDHExp; $s.records$ = [];
    $s.preim$ = Map2PointPreImages; $s.nmax$ = $n_{\max}$
  def sampleY($s$): return $(s.sSDH)$.sampleY();
  def isValid($X$): return $(s.sSDH)$.isValid($X$);
  def sampleH1($s$):
    $i \leftarrow 0$;
    $G \leftarrow (s.sSDH)$.sampleH1();
    do while (i < 2):
      sample $r_i \leftarrow_R \mathbb{F}_p$;
      preimageList = $(s.preim)(G^{r_i})$;
      sample $m \leftarrow_R \{0 \ldots (s.n_{\max} - 1)\}$;
      if len(preimageList) > $m$:
        $h_i \leftarrow$preimageList[$m$]; $i$+ = 1;
    $h \leftarrow (h_0, h_1); r \leftarrow (r_0 + r_1)$;
    if $h$ in $s.records$ abort("H1 collision");
    if $r = 0$: abort("Neutral element returned");
    $s.records$.append($r, G^r, h$); return $h$;
  def corrupt($s, h, Y$):
    if there is $(r, G, h)$ in $s$.records:
      return $(s.sSDH)$.corrupt($G, Y^{1/r}$);
  def DDH($s$,$h, Y, X, K$):
    if there is $(r, G, h)$ in $s$.records:
      return $(s.sSDH)$.DDH($G, Y, X, K^{1/r}$);

# Chaining the experiment objects for ristretto and decaf
sSdh = sSDH(sCDH);
coffeeExp = mapTwice_sSDH(Map2PointPreImages, $n_{\max}$,sSdh);
```

Figure 11: Experiment class definition mapTwice_**sSDH where** H_1 **hashes to** $\mathbb{F}_q \times \mathbb{F}_q$**.**

In order to avoid this purely technical complexity in our presentation we decided to analyze security of CPace here in the more complex setting where no ordering is enforced. For the purpose of the analysis, we thus had to modify the protocol from [Haa20] such that ordered concatenation (written oc(A,B)) is used for generating hash function inputs instead of just putting the initiator's message first.

We would like to stress that in case that the protocol control flow guarantees a defined sequence, as is e.g. the case for protocols such as TLS, there is no need security-wise to enforce use of ordered concatenation.

D CHAINING CHALLENGE GENERATOR CLASSES

Two different types of experiments can be distinguished. Firstly, the sCDH and the sCDH_sTCDH classes (Fig. 4 and Fig. 8) share the same API and specify two variants of the conventional CDH problem. Instances will be assigned a base point B in their constructor call, produce two elements from a prime-order group and answer DDH queries. Secret exponents will be revealed in case of corruption events. Also the objects inform the caller on which objects are accepted by their DDH oracle.

This API is used by the objects representing variants of the simultaneous Diffie-Hellman assumption. As security of all variants is ultimately based on the prime-order version of the sSDH problem all calls from $\mathcal{S}$ will ultimately be converted into a call to an instance of the sSDH class from Fig. 4. This is implemented by chaining the different classes.

As root of the chain first an instance of the prime-order sSDH class is generated which is parametrized by one of the two conventional CDH classes (sCDH or sCDH_sTCDH) by a constructor parameter. In its constructor the sSDH object will sample a generator B and create an instance of the sCDH or sCDH_sTCDH class, which will become a member of the sSDH root class. The base point is passed to this member object in its constructor call.

The challenge generator classes for the *simultaneous* problem variants will all produce base points for the simultaneous Diffie-Hellman problem through their API for producing H_1 samples. However the encoding of the base point varies. In the root class sSDH a uniformly sampled group element is returned directly. The derived classes could return the base point also in form of an encoding h that needs to be passed to the mapping construction for decoding.

We describe the chaining approach by using the example of single-coordinate protocol variants on curves with cofactor and twist security using a "map-once" primitive (our recommendation for twist secure Montgomery curves from Appendix F). The corresponding code is found in Fig. 8 below the class definition.

As all Montgomery curves having real-world relevance today come with twist security, the sSDH root object is instantiated with an sCDH_sTCDH object and we obtain an "sSdhExp" instance of the sSDH class. As the curve has a cofactor, "sSdhExp" will be passed to an instance "ccExp" of the class cofactorClearer and stored there as a member. "ccExp" makes sure that inputs for all queries, notably parameters 2,3,4 of the DDH function, will be mapped to the prime-order subgroup and transformed into calls to "sSdhExp". The "ccExp" object itself is then passed to the constructor of the distribution class $\mathcal{D}_\mathcal{G}$_sSDH which will be given a function reference to the implementation of the preimage calculator function for the map. This "distExp" object will return H_1 samples that need to be decoded using the Map2Point function in order to obtain group elements and require this encoding in its first operand in its DDH function. Finally an instance "twistSecExp" of the moduloNegationAdapter adapter class is created, having the "distExp" object as a member in its body.

Calls to the DDH method of the "twistSecExp" object will receive the base point h in a form that needs to be passed to Map2Point for decoding to a group element. The missing coordinate is reconstructed for the second, third and fourth operand of the DDH function and for both candidate points calls to the DDH method of the "distExp" instance will be issued. The latter will convert the base point query h to a group element and forward the query to the cofactor clearer. The latter will make sure that queries for points will have their low-order component cleared and issue a call to the sSDH root object which itself will forward the query to the sCDH_sTCDH instance in its body which uses its secret exponents for giving the response.

E NOTE ON SAMPLING OF SCALARS

The CPace protocol needs an algorithm for sampling scalars y for use as private Diffie-Hellman exponents. Ideally for a (sub-)group of order p, these should be sampled from a uniform distribution from $\mathbb{F}_p \setminus 0$. Sometimes, in particular if p is very close to a power of two $p = 2^l + p_0$ with $p_0 \ll p$, existing Diffie-Hellman libraries draw scalars from the set $\{1, \ldots, (2^l - 1)\}$ instead or include other structure in the scalars. Examples include the X25519 Diffie-Hellman protocol [Ber06] on Curve25519 with it's so-called scalar-clamping procedure which has been re-used for the X448 protocol on Curve448. We observed other libraries to fix some scalar bits to a defined 1 value in order to guarantee for a constant execution time in window-based scalar multiplication strategy. There are also good reasons for avoiding any additional calculation involving secret scalars y, such as might be required during rejection sampling, since any operation on y might put the secret at risk, e.g. due to side-channels. As CPace aims at being suitable for re-using existing well-tested libraries as-is, the implications for the protocol security due to non-uniform sampling of scalars need to be analyzed. Unlike for conventional Diffie-Hellman protocols, for CPace not only the confidentiality of ephemeral session keys might be affected but also information on the passwords could be leaked.

In order to consider this, one would introduce an additional game $\mathbf{G}_0$ b after the real-world setting and only change the scalar-sampling algorithm. In this game, we replace the possibly slightly non-uniform distribution of secret scalars y_a from honest parties in the real world with a fully uniform distribution. For a group order $c \cdot (2^l + p_0)$ implementations in the real world honest parties might draw the scalars y_a and y_b from $\{1, \ldots, (2^l - 1)\}$ instead of $\{1, \ldots, (p - 1)\}$. Let $\mathbf{Adv}^{\mathrm{DUN}}$ be the advantage of distinguishing $Y = B^y$ derived from a uniform or slightly nonuniform distribution of secret scalars y. Let l_1 denote the number of generated public keys Y_a, Y_b from honest parties and let l_2 the number of adaptive corruptions that reveal the secret exponents.

$$| \Pr[\mathbf{G}_0] \text{ - } \Pr[\mathbf{G}_0 \text{ b}] | \leq (p_0/p) \cdot (l_1 \cdot \mathbf{Adv}^{\mathrm{DUN}} + l_2)$$

As a result of our assessment the structure introduced by the "scalar clamping" defined for X25519 and X448 does not introduce a critical non-uniformity for CPace.

F RECOMMENDATIONS FOR ACTUAL INSTANTIATIONS OF CPACE

CPace was designed for suitability with different styles of Diffie-Hellman protocols on elliptic curves. Our perception is that it is possible to distinguish three main application scenarios with real-world relevance.

F.1 Short-Weierstrass

The first elliptic curves being standardized used a short-Weierstrass form. In practice only prime-order curves became of more wide-spread relevance. While some applications use point-compression, historically mostly a full coordinate representation is used for the network communication, (possibly for patent circumvention reasons that applied at the time). A typical example would be ECC on NIST-P256 and Brainpool curves [DSS13, ECC18, LM10].

In this ecosystem, we recommend to instantiate CPace using the following features

- Encode points Y_a and Y_b using full coordinates and use conventional point verification.
- Still, we recommend to actually only use the x-coordinate of the Diffie-Hellman result K for the session key, as TLS does so and some libraries use an x-coordinate-only ladder internally.
- We recommend to use the *nonuniform* Encode2Curve algorithms from the Hash2Curve draft, i.e. the "map-once" primitive, specifically because otherwise single-coordinate scalar multiplication strategies would not be practical for CPace. The point addition required by the "map-twice-and-add" approach would require full group operations. As mapping algorithm we recommend simplified SWU [FHSS$^+$19] because according to our analysis this is the least complex and most efficient variant working for the established curve set, notably NIST-P256.
- In this ecosystems, implementers should carefully evaluate the process of scalar sampling chosen in their library, in particular for the NIST-P256 and Brainpool curves. We observed that some libraries might not sample scalars sufficiently uniformly. Here we recommend rejection sampling.

For this instantiation the assumption set is modeled by the challenge-generator class "singleCoorExp" in Fig. 7.

F.2 Montgomery curve ladders

In addition to the established Weierstrass ecosystem, recently constructions based on Montgomery and (twisted) Edwards curves emerged. We believe that these designs became especially attractive as their design already considered typical implementation pitfalls from the very beginning. The wide-spread use of the prominent representatives X25519 and X448 [LHT16] resulted in standardization for internet protocols and by standardization bodies such as NIST.

The first implementations for CPace did focus on this type of Diffie-Hellman primitives. For the twist-secure Curve25519 and Ed449 we recommend the following configuration.

- Only use the u-coordinate on the Montgomery curve.
- Do not verify the curve equation explicitly, but check the neutral elements (all elements encoded with zeros), i.e. use the twist security. This way ScalarMultVfy and ScalarMult become the same function and X25519(X,y) and X448(X, y) could directly be used for this purpose. Note that checking for the neutral elements (all zero encoding for X448 and X25519) is absolutely mandatory for CPace. Here the discussion from [LHT16] which describes this check as optional does not (!) apply.
- For the map, we recommend to use non-uniform Elligator2 without co-factor clearing as this spares a field inversion (or alternatively spares a montgomery ladder implementation working on projective-coordinate inputs instead of an affine input.).

As the group orders are very close to a power of two, the clamped scalar method can be considered suitable also for CPace instantiations. For the construction described in the this paragraph the assumption set is modeled by the challenge-generator class "twist-SecExp" from Fig. 8.

F.3 Group abstraction

The speed advantage and the simplicity provided by the efficient and complete addition formulas available for Montgomery and (twisted) Edwards curves could only be obtained at the cost that protocols then need to deal with the co-factors. Not all protocols are, such as CPace, analyzed regarding this aspect. In order to allow for “the best of both worlds” projects are emerging which provide prime-order group abstractions on these curves, such as ristretto25519 and decaf448 [Ham15a, dVGT$^+$20]. As these abstractions work with the most efficient addition formulas currently known for elliptic curves, we do not see any incentive or advantage for single-coordinate instantiations for CPace. (In our opinion tightly constrained applications might rather opt for the Montgomery curve ladders anyway.) The designers of these abstraction frameworks also include mapping algorithms (“batteries included”) and enforce invalid-curve checks by their API design. Here we recommend the following parameters for CPace:

- Use the compressed point encoding from the abstraction.
- Use the built-in map-twice-and-add construction that comes “batteries included” with the abstraction.
- In our opinion, the verification check for the identity element should receive some attention. Comparisons should be carried out using the unambiguous encoding provided by the abstraction and *not* using the intermediate representation format employed internally for the Hisil-Wong-Carter-Dawson addition formulas [HWCD08].

As the group orders for ristretto25519 and decaf448 are very close to a power of two, using random-number generator outputs without rejection sampling can be considered suitable for deriving secret exponents for CPace. For this instantiation the assumption set is modeled by the challenge-generator class "coffeeExp" in Fig. 11.